

AD-A032 571

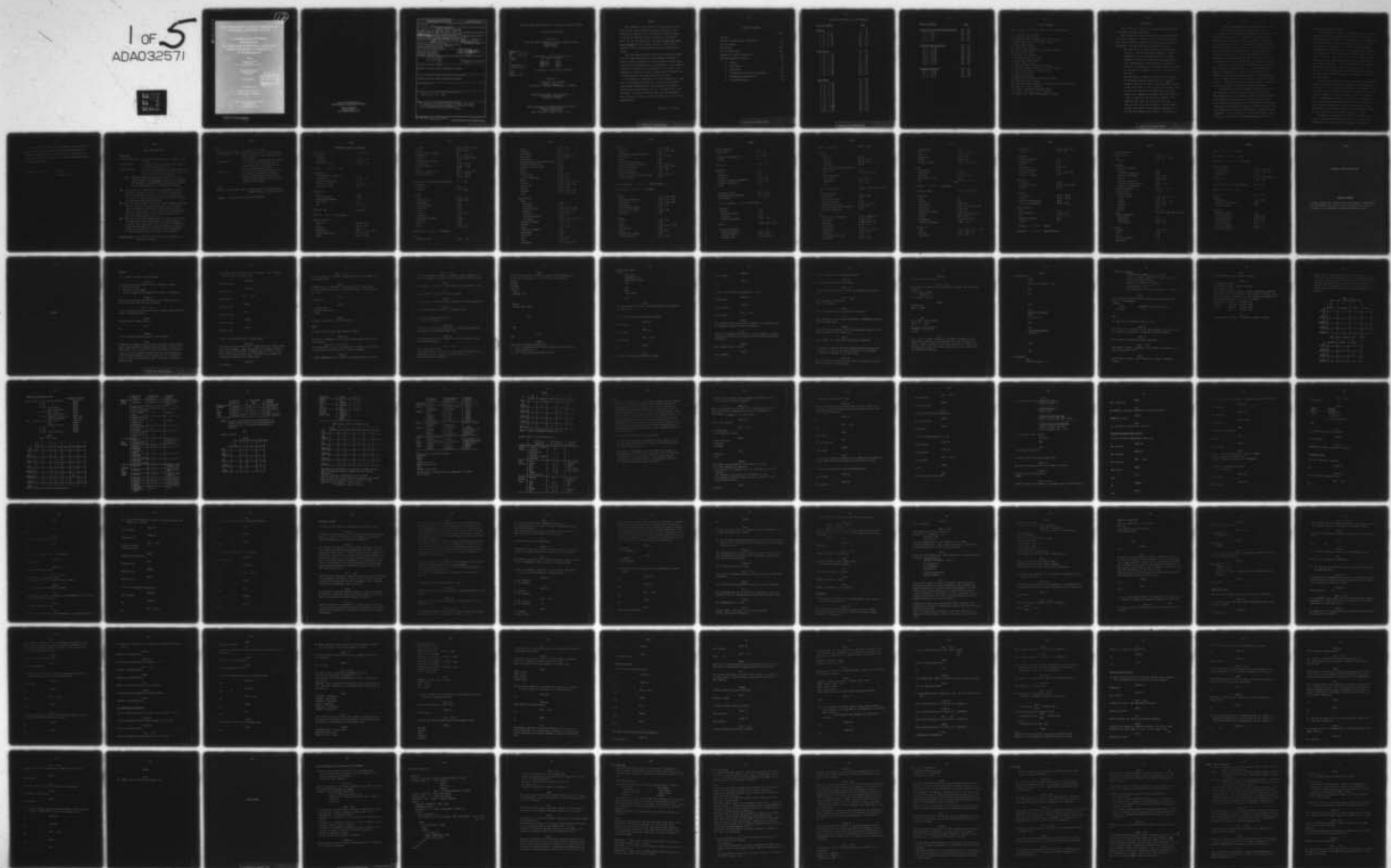
PURDUE UNIV LAFAYETTE IND PURDUE LAB FOR APPLIED IND--ETC F/G 9/2  
A LANGUAGE COMPARISON. A COMPARISON OF THE PROPERTIES OF THE PR--ETC(U)  
OCT 76 R ROESSLER, K SCHENK

N00014-76-C-0732

UNCLASSIFIED

NL

1 of 5  
ADA032571



AD A032571

FG

12

# THE INTERNATIONAL PURDUE WORKSHOP ON INDUSTRIAL COMPUTER SYSTEMS

## A LANGUAGE COMPARISON DEVELOPED BY THE LONG TERM PROCEDURAL LANGUAGES COMMITTEE-EUROPE COMMITTEE TC-3 OF PURDUE EUROPE

Edited by

R. Roessler and K. Schenk  
Physics Institute  
University of Erlangen-Nuernberg, W. Germany

Published with the support of  
The European Communities  
Brussels, Belgium

October 1976



Republished for

Department of the Navy  
Office of Naval Research

Purdue Laboratory for Applied Industrial Control  
Schools of Engineering  
Purdue University  
West Lafayette, Indiana 47907

### DISTRIBUTION STATEMENT A

Approved for public release;  
Distribution Unlimited



This Report is Published as Part of  
Engineering Experiment Station Bulletin 143 Series

Schools of Engineering  
Purdue University  
West Lafayette, Indiana 47907

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NR 049-388	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A LANGUAGE COMPARISON. A comparison of the Properties of the Programming Languages ALGOL 68, CAMAC-IML, CORAL 66, PAS 1, PEARL, PL/1, PROCOL, RTL/2 In Relation to Real Time PROGRAMMING.		5. TYPE OF REPORT & PERIOD COVERED
6. AUTHOR(s) Developed by The Long Term Procedural Languages Committee-Europe, Committee TC-3, Purdue Europe and Edited by R. Roessler and K. Schenk		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue Laboratory for Applied Industrial Control Schools of Engineering, Purdue University West Lafayette, Indiana 47907		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0732 NEW
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 440p		12. REPORT DATE October 1976
		13. NUMBER OF PAGES 462
		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)  Approved for public release; distribution unlimited.		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)  Please see pp. xix - xxix.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  A comparison of the key features of eight well known real time programming languages in relation to their use for industrial computer systems.		

DD FORM 1473

1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408244LB

INTERNATIONAL PURDUE WORKSHOP ON INDUSTRIAL COMPUTER SYSTEMS

A LANGUAGE COMPARISON

Developed by

THE LONG TERM PROCEDURAL LANGUAGES COMMITTEE-EUROPE  
COMMITTEE TC-3  
PURDUE EUROPE

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DOC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

A Comparison of the Properties of the  
Programming Languages

ALGOL 68	PEARL
CAMAC-IML	PL/1
CORAL 66	PROCOL
PAS 1	RTL/2

in relation to Real Time Programming

Edited by

R. Roessler and K. Schenk  
Physics Institute  
University of Erlangen-Nuernberg, W. Germany

Originally Published with the Support of  
The European Communities  
Brussels, Belgium

Purdue Laboratory for Applied Industrial Control  
Schools of Engineering  
Purdue University  
West Lafayette, Indiana 47907, U.S.A.



## PREFACE

This document is the result of a major effort of the LTPL-E (Long Term Procedural Languages Committee of the Purdue Europe Region of the International Purdue Workshop on Industrial Computer Systems) Committee to establish a basis for the specification of the LTPL or Long Term Procedural Language of the Workshop through a "meld of best features" of the several already existing a proposed languages.

The resulting study was carried out by getting a large group of people experienced in the languages involved to answer a series of questions which developed the main points concerning each language and its capabilities. The completed collection of answers, this document, will be used by the Committee to develop the best collective set of features for incorporation into the specification of the LTPL.

The Workshop is indebted to the LTPL Committee for their work in assembling this material. We are also grateful to the European Communities for financing much of the work involved and for publishing the first version of this work in Europe. We wish to thank the Office of Naval Research for providing the funds to allow its publication also in the United States.

Theodore J. Williams

TABLE OF CONTENTS

	Page
Preface . . . . .	v
Listing of Questions vs. Page Numbers . . . . .	ix
List of Authors . . . . .	xii
Introduction. . . . .	xiii
List of Definitions . . . . .	xvii
Alphabetical Table of Key-Words . . . . .	xix
Language Comparison Studies . . . . .	xix
1. Tasking. . . . .	1
2. Input/Output . . . . .	61
3. Algorithmic. . . . .	151
4. Configuration and System Description . . . . .	279
5. Implementation Experience. . . . .	325
6. Language Experience. . . . .	393

LISTING OF QUESTIONS VS. PAGE NUMBERS

<u>Question Numbers</u>	<u>Page</u>
<u>Tasking</u>	1 - 59
T 1 - T 10	3 - 14
T 11 - T 20	15 - 30
T 21 - T 30	30 - 38
T 31 - T 40	38 - 48
T 41 - T 50	48 - 56
T 51 - T 53	56 - 59
 <u>Input/Output</u>	 61 - 149
I 1 - I 10	63 - 83
I 11 - I 20	83 - 94
I 21 - I 30	94 - 104
I 31 - I 40	104 - 114
I 41 - I 50	114 - 122
I 51 - I 60	122 - 128
I 61 - I 70	128 - 136
I 71 - I 80	136 - 144
I 81 - I 86	144 - 149
 <u>Algorithmic</u>	 151 - 278
A 1 - A 10	153 - 168
A 11 - A 20	168 - 182
A 21 - A 30	182 - 193
A 31 - A 40	194 - 204
A 41 - A 50	204 - 212
A 51 - A 60	212 - 222
A 61 - A 70	222 - 232
A 71 - A 80	232 - 242
A 81 - A 90	242 - 256
A 91 - A 100	256 - 266
A 101 - A 110	267 - 276
A 111 - A 112	276 - 278



<u>Question Numbers</u>	<u>Page</u>
<u>Configuration and System Description</u>	279 - 323
C 1 - C 10	281 - 295
C 11 - C 20	295 - 307
C 21 - C 30	307 - 321
C 31 - C 32	321 - 323
 <u>Implementation Experience</u>	 325 - 391
IM 1 - IM 10	327 - 340
IM 11 - IM 20	341 - 350
IM 21 - IM 30	350 - 357
IM 31 - IM 40	358 - 366
IM 41 - IM 50	366 - 378
IM 51 - IM 60	378 - 388
IM 61 - IM 62	388 - 391
 <u>Language Experience</u>	 393 - 429
LE 1 - LE 10	395 - 409
LE 11 - LE 20	409 - 424
LE 21 - LE 25	424 - 429

# LIST OF AUTHORS

The following experts have contributed to this document:

J.G.P. Barnes, ICI, England  
A.F. Chalmers, GEC, England  
B. Eichenauer, ESG, Germany, now at GPP, Germany  
P. Elzer, University of Erlangen, Germany  
J. Heger, BBC, Germany  
M. Inderst, ESG, Germany  
N.V. Jones, GEC, England, now at SYSCON, England  
W. Kneis, GfK, Germany  
K. Kreuter, ESG, Germany  
G. Louit, S.T.E.R.I.A, France  
M. Malagardis, IRIA, France  
G. Michel, IRIA, France  
K. Pelz, University of Erlangen, Germany  
G. Petitalot, CERCI, France, now at ESRO, Germany  
I.C. Pyle, University of York, England  
J. Reed, GEC, England  
J. Robert, CAP, France  
R. Roessler, University of Erlangen, Germany  
N. Scheuffele, ESG, Germany  
H.J. Schneider, BBC, Germany, now at University of Stuttgart, Germany  
D.N. Shorter, B.S.C., England  
G. Verroust, Université D'Orsay, France  
I.C. Wand, University of York, England  
J.T. Webb, Royal Radar Establishment, England

## INTRODUCTION

This is the first issue of a reasonably complete version of the LTPL-E language comparison.

Some people will say that they would not have expected to ever see it completed. Well, it is true that it took a bit more time to finish the document than the group had expected beforehand. But there were several reasons for this:

- The main one is that the amount of work grew while doing it. This was because the number of candidate languages had to be increased from five to eight. Likewise the number of questions to be answered for each language had a strong tendency for expansion. But this also means that the document finally produced is now much more complete and gives a better survey of the field.
- Another reason is that all the work had to be done by volunteer committee members who did it mostly in their spare time as left by their principal professional activities, or even in their free time at home. But this also means, on the other hand, that these people did a tremendous job under difficult conditions. I want to thank all contributors on behalf of the committee very much for their effort and work. It does not seem to me that much has to be said about purpose and context of the language comparison. It has been described several times in the Minutes of



of the "Purdue Workshop" and also externally at the IFAC Seminar on Realtime Programming in 1974.

According to the plans described there, the document originally was intended only to serve as a data base for the further work of the LTPL-E group, i.e. for the development of a standard proposal for a programming language for real-time applications. Thus the character of the book is nearly that of a committee internal paper. It is restricted to be an extensive collection of facts and material for further evaluation and deliberately does not contain any conclusions concerning the relative qualities of the compared languages.

This may be looked upon as a deficiency but, comparing this study with other ones done, e.g., by software-houses, where one language had to "win", I am convinced that an unbiased, though detailed investigation of the possibilities of the various languages can be of great value to interested experts. LTPL-E will work out separate studies on evaluation criteria and sample problems for further comparisons.

The committee-internal character of the study also was the reason for another, maybe more serious restriction: The comparison does not include any and all possible programming languages which might be used for realtime purposes. We included the languages which were developed by LTPL member organizations and which are supported in such a way that they are intended to influence the forthcoming standards proposal remarkably. But as time goes on, this situation may change in so far as additional language development groups join

LTPL--as has already happened.

But despite this there has already been so much interest expressed from the outside that we decided to make the language comparison available to a broader public. The necessary editorial work for this could only be done with the support of the European Communities. In order to make the paper more readable for the outside user we have tried to structure it slightly and we have produced a "two-level" table of contents. The first level is a table of the principal key-words, (pp. xix - xxix) arranged alphabetically and referenced to the respective chapters. On the second level these chapters are mapped onto the page numbers of the book (pp. ix - x). This split has been made because of the possibility that the language comparison may be expanded or changed and so will the page numbers.

Another remark should be made for the sake of the external user concerning the chapters on language- and implementation experience: At the time of the editing of the document it was still difficult to answer the respective questions in a consistent way, because the various languages were in a different state of implementation and use. But as it is intended to use a computerized text-handling system for easier adaption of the language comparison to necessary changes and future development, the reader can expect to see an updated version of these chapters as soon as new material is at hand.

I therefore hope that this book will be of some use to experts outside of the LTPL-group and I may say that the committee will be grateful for any comment and feedback.

Finally let me thank all of the experts who have contributed for their effort and all LTPL-member organizations for sending and supporting these experts which in fact means supporting our group and the idea of a common programming language for realtime purposes.

Erlangen, July 1975

P. Elzer

Chairman LTPL-E

## LIST OF DEFINITIONS

### Happenings

- connected happening - an occurrence generated as a direct effect of a PA
- free happening - an occurrence not generated as an effect of a PA but of interest to it
- loose happening - an occurrence generated as a side-effect of a PA, e.g. end-of-file, overflow. This corresponds to the PL/I 'ON' condition.

Note: Messages which are generated by one PA for consumption by another are connected to the first, and free to the second. Only external interrupts are free to all PAs (and generally of interest to only one).

LAM - LAM stands for "Look-At-Me" and is the CAMAC way of describing an external stimulus to a computer system which might, for example, be used to give us an interrupt, but may also in an appropriate system design be used to trigger an autonomous input/output transfer.

MNQ - MNQ. This denotes one of the block transfer modes and stands for multidevice action, repeat while not Q. This means apply a command to a sequence of CAMAC modules, each of which can respond with a signal known as Q. This continues while the responses are  $Q = \emptyset$ , but is terminated after the first module which returns  $Q = 1$ .

MAD - MAD. This is the IML nomenclature for a basic type of block transfer specified in CAMAC where it is called address scan mode. The idea here is that a sequence of CAMAC addresses are accessed and the decision whether to continue scanning the sequence or terminate is again determined by the Q-response, when the operation is performed.

Multiprocessor - a collection of processors, each capable of executing a segment.



PAs

parallel activity (PA) - the execution of a PCE

synchronization of PAs - controlling PAs so as to maintain some desired sequential relationship between their effects

independent PA - a PA (first) is independent from another PA (second) which has initiated the first, when the first might continue after the second has finished

dependent PA - a PA (first) is dependent from another PA (second) which has initiated it, when the termination of the first is automatically connected with the termination of the second.

PCEs

parallel code element (PCE) - segments which could be executed simultaneously by a multiprocessor.

Segment - a sequence of executable statements.

ALPHABETIC TABLE OF KEY-WORDS

Algorithmic	A 1 - A 112
Allocation	
Devices	I 19 - I 22
Resources	A 105
Resources to PAs	T 26 - T 30
Arrays	----- Data
Assignments	
Assignments by Pointers	A 54
Conversion	A 70, A 71
Kinds of Assignments	A 79
Priority to PAs	T 21, T22
Resources to PAs	T 25
Attributes	
Data	A 49
Data Structures	A 56
Hardware Environment	C 13
Identifiers	A 53
PAs	C 15, T 8
Character Set	A4, IM 22
Commands	----- Statements
Compiler Aspects	
Capacity	IM 10, IM 11
Code	IM 27, IM 28
Computer	IM 8
Diagnostics	IM 49 - IM 51, IM 54
Language Experience	LE 8 - LE 11
Layout	IM 22 - IM 26

Linkage	IM 32, IM 33, IM 35
Macros	IM 20, IM 21
Minimum Unit to Compile	T 3
Modularity	IM 12 - IM 16
Overlay	IM 29, IM 39, IM 40
Parameterized Compiler	C 26
Portability	IM 58 - IM 62
Program Handling	A 104 - A 109
Size	IM 9
Time for Compilation	IM 11, IM 30, IM 31
Writing Aspects	IM 17 - IM 19, IM 58 - IM 61
Configuration and System Description	C 1 - C 32
Constants	
Conversion	A 18
Notation	A 17, A 20
Control	
Date	A 26, I 83
Flow-Control	A 80 - A 82
Happenings	T 33
I/O Operations	A 83
Interrupts	T 33
Layout	I 25
Priority	T 23, T 24
Procedures	A 83
Processing Language	A 104
Resources	T 28
Tasks	A 83
Time	A 26, I 83
Coroutines	----- Procedures
Data	
Access to Data	A 52 - A 65

Arrays	A 35 - A 38
Attributes	A 49
Conversion	A 66 - A 71
Declaration	A 7 - A 16, A 51
Destruction	A 75, A 76
Exchange between External Devices	I 85
Exchange between PAs	T 16
Extension of Data Modes	A 11Ø
Files	I 78
Graphic	I 49
Indexed Data	A 35 - A 38
I/O of Binary Data	I 56, I 57
Modes	A 17 - A 34
Process	I 61- I 63, I 71, I 72
Structures	A 39 - A 47, A 63
Tables	A 35 - A 38
Transfer	A 72 - A 74, I 86
Types	A 17 - A 34, A 48
Declarations	
Data	
Constants	A 2Ø
Data Sets	I 73
Data Structures	A 36 - A 38, A 4Ø
General	A 7 - A 11, A 14
Pointers	A 33, A 34
Time-Information	A 26
Variables	A 15, A 16, A 22, A 32
Declare Statement	A 13, A 14
Devices	I 12 - I 14
Files	I 73, I 81
Function Procedures	A 92
Happenings	T 31
Key-words	A 49
PAs	T 13 - T 15
PCEs	T 6, T 9
Procedures	A 91, A 92, T 9



## Devices

Allocation	I 19 - I 22
Classes	I 11, I 56, I 60
Data Exchange between Devices	I 85
Description	I 12 - I 18
Graphic	I 50
Interrupts by Devices	I 67
Linkage Software - Hardware	C 4 - C 8, C 11
Process Peripherals	I 60, I 65, I 66
Safety Measures	I 39, I 40
Status Information	I 44
Synchronization of Device Access	I 32, I 34, I 51
Virtual Devices	I 81

Error Mechanisms ----- Safety Aspects

Expressions ----- Statements

## Files

Access	I 75, I 79, I 80
Addressing Mechanism	I 76, I 77, I 79
Data Types	I 73, I 78, I 80
Declaration	I 73, I 81
File-Handling System	C 23
Operations on Files	I 74, I 80
Protection	I 36

## Formats

Control	I 25, I 27
Files	I 76
General	I 23, I 25 - I 30
Process Data	I 63
Syntax	I 9, I 10
Unformatted Transfer	I 86
Variable Formats	A 25, I 24

General Questions

Algorithmic	A 1 - A 3
I/O	I 1 - I 5
Language Implementation	IM 2 - IM 7
Tasking	T 1

Graphic I/O	I 48 - I 55
-------------	-------------

Happenings

Control	T 33
Definition	T 31
Examples	T 31, T 32
Hardware or System Happenings	C 9 - C 12
Software Happenings	C 18, C 19
Types	T 31

Hierarchy of PAs	T 34 - T 37
Implementation Experience	IM 1 - IM 62
Input/Output	I 1 - I 86

I/O - Instruction ----- Statements

Interrupts

Control	T 33
External Devices	I 67
I/O Error	I 45
Process Interrupts	I 68 - I 70

Labels

A 50, A 80 - A 82
-------------------

Language Experience

Language Designer	LE 1 - LE 7
Compiler Writer	LE 8 - LE 11
Language User	LE 12 - LE 23

Library Facilities

IM 55 - IM 57

Linkage

Editing

Capacity

IM 38

General

IM 32 - IM 37

Overlays

IM 39, IM 40

Real Time

IM 36

Problem Software-System Software

Happenings

C 19

PAs

C 15, C 16, C 18

Software-Hardware

Environment

C 13, C 14

Happenings

C 9 - C 12

I/O

C 1 - C 8

Macro Instructions

A 102, A 103, IM 20, IM 21

Operating System

Generation

C 20 - C 24

Initiation of PAs

T 2

On-Line Updating

C 27

Organization of PAs

C 15, C 16, C 18

Producing Application Software

C 25, C 26

Software Happenings

C 19

PAs

Allocation of Resources

T 25 - T 30, T 51

Attributes

C 18, T 12

Commands

T 40, T 41, T 46

Declaration

T 13 - T 15

Description

C 16

Happenings

T 7, T 31 - T 33

Hierarchy

T 34 - T 37

I/O Operation as a PA

I 31

Operations

T 38, T 39

Organization	C 15
PCEs	T 6, T 17 - T 19
Priority	T 20 - T 24
Scheduling	T 46, T 47
States	T 10, T 52
Synchronization	T 49 - T 51, T 53
PCEs	
Declaration	T 9
Definition	T 4, T 5
Execution	T 7, T 8, T 32
Operations on PCEs	T 38
PA	T 6, T 17 - T 19
Pointers	-----
References	
Priority of PAs	T 20 - T 24, T 36
Procedures	
Call	T9
Control	A 83
Conversion	A 68, A 69
Declaration	T 29, A 91, A 92, A 101
Definition	A 90, A 99
Function Procedures	A 92, A 97
General	A 90, A 95, A 96, A 97
Library	IM 55, IM 57
Parameters	A 93, A 94, A 98
Standard Procedures	A 100, A 101
Process I/O	
Data	I 61 - I 63, I 71 - I 72
General	I 58 - I 60, I 64
Interrupts	I 67 - I 70



Peripherals	I 60, I 65, I 66
Transfer	I 71, I 72
References	
Access to Pointers	A 65
Dereferencing	A 54
Library	A 6
Linked List	A 46, A 47
Pointer-Variables	A 43
Pointer to Data	A 33, A 44, A 45
Pointer to Pointer	A 34
Resources	
Allocation	A 105
Resources of PAs	C 14, T 25 - T 30
Control	T 28
Run-Time	
Diagnostic	IM 52 - IM 54
General	IM 44 - IM 48
Maximum Configuration	IM 43
Minimum Configuration	IM 41, IM 42
Safety Aspects	
Data	A 5
Error-Handling	I 42 - I 47
I/O	I 35 - I 41
Language Security	LE 17
Program	A 5
Scheduling	----- Timing
Semaphores	----- Synchronization

## Special Questions

Algorithmic	A 110 - A 112
I/O	I 64, I 82 - I 87
Tasking	T 2

## Statements

### Compound Statements

Blocks	A 89
Conditional Execution	A 86
Iterative Execution	A 87, A 88
Macros	A 102, A 103
Parallel Execution	A 85
Procedures, Subroutines	A 90 - A 101
Sequential Execution	A 84

Declare Statement	A 7, A 13, A 14
-------------------	-----------------

Extensibility	A 110
---------------	-------

### I/O-Instructions

Device Access	I 10, I 12, I 16
Files	I 74, I 81
Graphic	I 48, I 52 - I 55
Process	I 59 - I 65
Safety	I 35
Syntax	I 6, I 9
PAs	T 21, T 23, T 24, T 40, T 41, T 46

### Simple Statements

Assignments	A 79
Expressions	A 77, A 78
Flow Control	A 80 - A 83

## States

Devices	I 34, I 44
Files	I 80
PAs	T 10
PA State Diagram	T 11
Resources	T 28

Status ----- States

Structures ----- Data

Subroutines ----- Procedures

#### Synchronization

Device-Access	I 32, I 33, I 34
I/O Operations	I 31, I 33, I 34
Mechanisms	T 49
PAs	T 33, T 49, T 50, T 53
Process I/O	I 68 - I 70

Syntactic Form of I/O-Instructions	I 6 - I 10
------------------------------------	------------

Tables ----- Data

Tasking	T 1 - T 53
---------	------------

#### Timing

Mechanism	T 43 - T 46
Scheduling Happenings	T 48
Scheduling PAs	T 47

#### Variables

Address Variable	A 32
Device-Variables	I 20, I 22
Format-Variables	A 25, I 24
Initialization	A 15, A 16
Label-Variables	A 81
Pointer-Variables	A 32, A 43
String-Variables	A 24

LANGUAGE COMPARISON STUDIES

Editor's Remark

Answers marked with (\*) were found insufficient or inconsistent by the Editors, but could not yet be updated due to lack of manpower in the respective language development groups.



-1-

TASKING

General

1- Is tasking included in the language?

ALGOL 68

A range of tasking facilities exist in ALGOL 68, namely:

1. Parallel clauses
2. Variables of mode sema
3. Operators level, up and down which operate on semaphores.

CORAL 66

Not language elements, the ability to use tasking features is given by incline code and macro features.

PAS 1 - PL/I

A set of tasking features exists in the language (some from PL/I subset and some from PAS 1).

PEARL

A large set of tasking features.

PL/I

Yes

PROCOL

A set of tasking features exists in the language.

RTL/2

Although the language does not contain explicit tasking statements, nevertheless it contains features (STACK variables, SVC DATA bricks, etc.) with which tasking mechanisms can be build.

A family of multi-tasking systems (MTS) has been developed in RTL/2 and these systems can be considered to be an extension of RTL/2. This questionnaire has been answered with respect to these systems.

2- Is there only one possibility for tasking - i.e. is there only one way to initiate a PA?

ALGOL 68

Yes, only one.

CORAL 66

Yes, only one.

PAS 1 - PL/I

Yes, only one.

PEARL

Yes, only one.

PL/I

Yes, only one.

PROCOL

Yes, only one.

RTL/2

Yes, only one.

3- What is the minimum unit of compilation?

ALGOL 68

The natural unit of compilation in ALGOL 68 is the serial clause (text between begin....end), there being no concept of the external module as it exists in FORTRAN. Section 2.3(c) of the Report indicates that ALGOL 68 is intended to be suitable for independent compilation of program parts.

CORAL 66

A procedure.

PAS 1 - PL/I

A PL/I procedure or one PAS 1 program containing the system part and problem part.

PEARL

A PEARL module - containing a "system-division" (description of the configuration) and/or a "problem-division" (the appropriate code).

PL/I

A procedure.

PROCOL

- A common part
- external subroutine
- a task.

RTL/2

A module - consisting of several bricks.

PCEs

4- What can be Parallel Code Elements (PCEs)?

ALGOL 68

Any of the statements in a collateral-void-clause can be a PCE.  
For example:

par (x:=1, y:=2, z:=3);

The statements x:=1 etc., are executed in parallel. Each element of a collateral-void-clause is a unitary statement.

CORAL 66

A CORAL SEGMENT which is a group of procedures within a thread.



PAS 1 - PL/I

Several independent tasks with subtasks. A PL/I procedure or single statements connected to an event within a PAS 1 program.

PEARL

A statement or collection of statements are declared as a task.

PL/I

Main procedures are parallel code elements.

PROCOL

A task can be a PCE and may have external or internal subroutines.

RTL/2

Any parameterless procedure is a potential PCE.

5- Are PCEs defined explicitly?

If so, give example.

ALGOL 68

No, they are not defined explicitly - although they could be procedure calls which would mean that.

CORAL 66

Not defined explicitly in CORAL but in the linkage mechanisms for system generation.

PAS 1 - PL/I

Not in the case of PL/I

main tasks and PAS 1 tasks but in the case of activation of PL/I subtasks by use of priority - or event-option, e.g. :

CALL UP PRIORITY (S) EVENT (E).

# PEARL

No, PCEs are implicitly defined together with the declaration of PAs or are assigned to a PA at the time of activation.

Example:

MODULE;

PROBLEM;

TASK TA1:

BEGIN

TASKNAME TA2

.

.

.

BEGIN;

ACTIVATE TA2: BEGIN

.

.

.

END

.

.

.

END;

# PL/I

Yes.

# PROCOL

Each task is defined explicitly by:

1. a set of parameters given conversationally at the generation of the system (S.G.)
2. a set of declarations and instructions.

PROCOL (continued)

Example:

```
TASK TRX10 ;  
SHORTEGER J, K;  
SUBROUTINE LIRCAR;  
PROC;  
.  
.  
END;  
MAIN;  
CALL LIRCAR;  
.  
.  
END;
```

RTL/2

Not defined explicitly but at system generation or through a procedure call.

6- Can PCEs be named and by what mechanism?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Not relevant.

PEARL

No.

PL/I

No, but it can be known by several names.

PROCOL

Not relevant.

RTL/2

No.

7- Can a user specify a residence for a PCE?

ALGOL 68

Not possible.

CORAL 66

Yes, possible.

PAS 1 ~ PL/I

Not possible.

PEARL

Yes, possible with limitations in order not to interfere with an automatic storage allocation mechanism.

PL/I

Not in the language. The only place in the language in which residency is specifically visible is in the FETCH and RELEASE statements.

PROCOL

Yes, possible (at S.G.time).

RTL/2

Yes, possible.



8- Is it possible to have reentrant PCEs?

ALGOL 68

No, only procedures may be reentrant.

CORAL 66

Can be reentrant, but they are not all necessarily defined as such.

PAS 1 - PL/I

PL/1 Subtasks are reentrant  
PAS/1 not reentrant.

PEARL

No, only procedures may be declared as reentrant.

PL/I

Yes, procedures can be declared RECURSIVE or REENTRANT with the obvious capabilities.

PROCOL

No, may be reactivated but won't be recommenced before its completion (tasks are not reentrant).

RTL/2

In principle, yes, since code is basically reentrant.

9- Is there a language difference between the declaration/execution of a PCE and the declaration/call of a procedure?  
Please give examples of all four cases.

ALGOL 68

Yes, any unitary statement may be a PCE. If the PCE is a procedure call there is no difference.

CORAL 66

No difference (S.G. time).

PAS 1 - PL/I

No difference between the declarations but between the executions.

Examples:

1. PCE: CALL P EVENT ( )  
          PRIORITY ( );
2. Procedure: CALL P;

PEARL

Declaration:

TASK T1: BEGIN

.

.

.

END;

DCL P PROC = [attributes]

BEGIN.....END;

---

ACTIVATE T1 [options];

CALL P;

PL/I

Yes, a PCE is a MAIN procedure. Only MAIN procedures can be activated as tasks and every task must have one such MAIN procedure. A MAIN procedure can only be activated as a task or by the operating system. All other procedures can be activated only as sequential procedures.

PROCOL

1. Declaration:

```
PCE
TASK
Subroutine <name> [(...)]
.
.
PROC;
.
.
END;
.
.
.
MAIN;
.
.
END;
External subroutine
*RET
Subroutine...
.
.
PROC;
.
.
END;
Internal subroutine
Subroutine
.
.
.
PROC;
.
.
END;
```

2. Execution:

```
PCE
ACTIVATE <name>; or
```

PROCOL (continued)

```
ACTIVATE <name> WHEN <t> (t:= I/O unit)
(if T1 = H, task: immediately excuted; if not, it
is queued (runnable state) )
external and internal subroutine
```

---

```
CALL <name> [(...)] ;
(the subroutine is accessal immediately if free).
```

RTL/2

Declarations both the same since PCE is a procedure but task execution via MAKETASK.

Example:

PCE	MAKETASK (stack, P, priority)
procedure	P ()

PAs

10- What are the possible states of a PA?

ALGOL 68

The states are not defined in the language Report, but they will be in effect: non-existent, suspended, running/runnable.

CORAL 66

Non-existent, suspended, running, scheduled.

PAS 1 - PL/I

The states: runnable, suspended PL/I and PAS 1, scheduled, running, non-existent PL/I.

PEARL

Non-existent, dormant, idle, scheduled, runnable, suspended, running.



PL/I

Non-existent, active, waiting, inactive.

PROCOL

For hardware tasks:

Non-existent, idle, runnable, running

for software tasks:

non-existent, idle, runnable, running, suspended, scheduled.

The states 'idle' and 'dormant' are the same in PROCOL: once a task is created, a segment and a data area are assigned to it.

The following functions not exist in PROCOL:

Exterminate	Delay	(another task)
Delete	Continue	(another task)
Kill me	Suspend	(another task)
	Kill	(another task)

RTL/2

Non-existent, scheduled, suspended, runnable, running.

- 11- Please write a PA-state-matrix giving each state a column and a row. Mark each transition which is possible from row to column. Please group the commands given above into "system actions/operations by a task on itself operations on another task" and draw up a table with three columns "functional operation/state change of object task/argument(s) required".

PAS 1 - PI/I

Task - state - matrix for PL/I tasks

To From	Ø non existent	4 suspended	5 runnable	6 running
Ø non existent			X	
4 suspended			X	
5 runnable				X
6 running	X	X	X	

Task - state - matrix for PAS 1 - tasks

To From	3 scheduled	5 runnable	6 running
3 scheduled		X	
5 runnable			X
6 running	X	X	

Transitions between Task states

		transition induced by
PL/I - Tasks:	$\emptyset \rightarrow 5$ : CALL XY PRIO (1 $\emptyset$ );	other task
	$5 \rightarrow 6$ : —	system
	$6 \rightarrow 5$ : —	system
	$6 \rightarrow \emptyset$ : STOP; RETURN;	itself
	normal termination	system
	$6 \rightarrow 4$ : WAIT (event);	itself
	REQUEST (semaphore);	itself
	$4 \rightarrow 5$ : COMPLETION (event)='1'B;	other task
	RELEASE (semaphore);	other task
PAS 1 - Tasks:	$3 \rightarrow 5$ : SET (event);	other task
	time	system
	external interrupt	system
	$5 \rightarrow 6$ : —	system
	$6 \rightarrow 5$ :	system
	$6 \rightarrow 3$ : normal termination	system

PEARL

Task - state- matrix

	$\emptyset$ non existent	1 dormant	2 idle	3 sched.	4 suspend.	5 runnable	6 running
$\emptyset$ non existent		X	X				
1 dormant	X			X		X	
2 idle	X			X		X	
3 scheduled		X	X			Y	
4 suspended		X	X	X		XY	
5 runnable		X	X	X	X		Y
6 running		X	X	X	X	Y	

Where "Y" means an automatical transition by O.S.

	FUNCTIONAL OPERATION	STATE-CHANGE OF TASK	ARGUMENTS REQUIRED
SYSTEM ACTIONS	DECLARATION	$\emptyset - 1$	task variable
		$\emptyset - 2$	task-constant + code
	END OF BLOCK	$1 - \emptyset, 2 - \emptyset$	
	actual activation of a scheduled task	$3 - 5$	schedule
	actual continu- ation of a sus- pended task (schedule)	$4 - 5$	schedule
	all resources available	$5 - 6$	
	at least one re- source not avai- lable	$6 - 5$	
SELF ACTIONS	DELAY or	$6 - 4$	intervall
	synchronization		sema, bolt
	SUSPEND	$6 - 4$	
	TERMINATE (schedule is valid)	$6 - 3$	
	TERMINATE (if task-constant)	$6 - 2$	
	TERMINATE (if task-variable)	$6 - 1$	
ACTIONS ON OTHER TASKS	ACTIVATE	$1 - 3$	schedule, task- variable, code
	ACTIVATE	$1 - 5$	task-variable, code
	ACTIVATE	$2 - 3$	schedule, task- constant
	ACTIVATE	$2 - 5$	task-constant
	PREVENT	$3 - 2$	task-constant
	PREVENT	$3 - 1$	task-variable
	TERMINATE (schedule is valid)	$4, 5 - 3$	task



	FUNCTIONAL OPERATION	STATE-CHANGE TASK	ARGUMENTS REQUIRED
ACTIONS ON OTHER TASKS	TERMINATE	4,5 - 2	task-constant
	TERMINATE	4,5 - 1	task-variable
	CONTINUE	4 - 5	task
	SUSPEND (DELAY)	5 - 4	task ( intervall)

Remark: All state changes mentioned under 'SELF ACTIONS' and 'ACTIONS ON OTHER TASKS' can also be performed by system automatically if the respective operations are scheduled.

PL/I

Not yet available.

#### PROCOL

Task - state- matrix for hardware tasks

From	To	Ø non existent	2 idle	5 runnable	6 running
Ø non existent			X		
2 idle				X	X
5 runnable					X
6 running			X		X

GENERATION	CREATE	∅ — 2
SYSTEM	RUN	6 — 5
ACTIONS	PREEMPT	5 — 6
SELF	MYSTATUS	
ACTIONS	EXIT	5 — ∅
ACTIONS	EXECUTE	2 — 6
ON OTHER	SCHEDULE	
(SOFT) TASKS	TERMINATE	
	SIGNAL	
	STATUS	

Task - state - matrix for software tasks

To From	∅ non existent	2 idle	3 scheduled	4 suspended	5 runnable	6 running
∅ non existent		X				
2 idle			X	X	X	
3 scheduled					X	
4 suspended					X	
5 runnable						X
6 running		X	X	X	X	

The states idle and dormant are not separated in PROCOL, because once a task has been created, a segment and a data area are assigned to it.

The other states are approximately the same in PROCOL.

From any state except 3, it is possible to come back to 2 by CANCEL.

From any state it is possible to go to 3 by EVERY or AFTER

State vector: 1<sup>st</sup> component : internal states

2<sup>nd</sup> component : momentary location.

	FUNCTIONAL OPERATION	STATE CHANGE OF OBJECT TASK	ARGUMENT REQUIRED
GENERATION	CREATE	$\emptyset \rightarrow 2$	$\langle \text{TASK} \rangle$
SYSTEM	CANCEL	$2, 5, 4, 6 \rightarrow 2 \text{ or } 3$	$\langle \text{TASK} \rangle$
ACTIONS	EXPERMINATE	can be done	$\langle \text{TASK} \rangle$
	READY	$3 \rightarrow 5$	$\langle \text{TASK} \rangle$
	RUN	$5 \rightarrow 6$	$\langle \text{TASK} \rangle$
	PREEMPT	$6 \rightarrow 5$	$\langle \text{TASK} \rangle$
	INHIBIT	$6 \rightarrow 4$	$\langle \text{TASK} \rangle$
	UNSUSPEND	$4 \rightarrow 5$	$\langle \text{TASK} \rangle$
SELF	SCHEDULEME		$\langle \text{TIME} \rangle / \langle \text{EVENT} \rangle$
ACTIONS	MYSTATUS	scheduled	
	EXIT	$6 \rightarrow 2 \text{ or } 3$	$\langle \text{TASK} \rangle$
	DELAYME	$6 \rightarrow 4$	$\langle /=/\text{TOPS} \rangle \langle \text{TIM (n)} \rangle$
	WAIT	$6 \rightarrow 4$	$\langle \text{EVENT} \rangle \langle \text{TOPS} \rangle$
	SECURE	$6 \rightarrow 4 \text{ or } 6$	$\langle \text{SEMAPHORE} \rangle$
ACTIONS	CANCEL (EXECUTE)	$2 \rightarrow 5$	
ON	SCHEDULE	$2 \rightarrow 3$	$\langle /=/\text{TOPS} \rangle \langle \text{TIM (n)} \rangle$
OTHER	CANCEL (TERMINATE)	$6, 5, 4 \rightarrow 2 \text{ or } 3$	$\langle \text{TASK} \rangle \langle \text{TIME} \rangle \langle \text{EVENT} \rangle$
SOFT	RELEASE	$4 \rightarrow 5$	$\langle \text{SEMAPHORE} \rangle$
TASKS	SIGNAL	$4 \rightarrow 5$	$\langle \text{EVENT} \rangle$
	STATUS	scheduled	$\langle \text{TASK} \rangle$

There are not the following functions:

EXTERMINATE

DELETE

KILLME

DELAY (another task)

CONTINUE (another task)

KILL

SUSPEND (another task)

The assignment of variable priority to other task or to itself is not allowed.

RTL/2

To From	Ø non existent	3 scheduled	4 suspended	5 runnable	6 running
Ø non existent		X			
3 scheduled	X			X	
4 suspended	X			X	
5 runnable	X		X		Y
6 running	X		X	Y	

Where "Y" means an automatical transition by O.S.

Tasking functions for RTL/2 MTS systems.

	Functional Operations	State change of object task	Arguments required
System actions	timeout i.e. delay	4 - 5	
	finished	5 - 6	
	processor etc. available	6 - 5	
	resources no longer available		
Self actions	STOP	6 - 4	task (itself)
	SECURE	6 - 4	semaphore
	WAIT	6 - 4	event
	DELAY	6 - 4	time interval
	KILL	6 - Ø	task (itself)
	KILLME end of the code	6 - Ø	
Actions on other atsks	MAKETASK	Ø - 3	stack, proc, priority
	START	3,4 - 5	task
	STOP	5 - 4	task
	KILL	3,4,5 - Ø	
	RELEASE	4 - 5	semaphore
	SET	4 - 5	event



Notes:

1. A task is runnable if it is both not suspended and not stopped. Some forms of suspension (DELAY, WAIT, SECURE) can only be self-imposed and so are mutually exclusive. A task may be STOPPED/STARTed by another task but if STARTed will only go into the runnable state if it is also not suspended by a DELAY, WAIT or SECURE. The suspended state in the diagram covers all existent tasks not actually runnable except for the case of newly created tasks which have been considered to be in the scheduled state until STARTed. In practice the scheduled state is identical to the suspended state (with suspension caused solely by STOP) except of course that the program counter has its initial value.
2. The idle and dormant states do not exist.
3. In small MTS systems the non-existent state does not exist and all tasks are effectively placed in the scheduled state at system generation time.

The operations MAKETASK, KILL and KILLME do not exist in such systems. The end-of-code operation (i.e. exit from main procedure) can be considered an error in these systems - the task is effectively placed in the scheduled state but any attempt to START it again is ignored.



12- What are the visible (and accessible) attributes of a PA  
(e.g. priority, state, residence)?

ALGOL 68

The only attribute of a PA that can be interrogated is the integer value of any semaphore that it may be dependent on.

For example:  $i := \text{level } s$ ;  $\phi s$  is a sema  $\phi$

CORAL 66

Not in the language.

PAS 1 - PL/I

- Its priority
- its termination (event)

PEARL

- Task identifier
- priority
- residence

PL/I

- Priority
- name

PROCOL

The visible (and accessible) attributes of a PA are:

- its type (hardware or software)
- its priority (for hardware task, the interrupt level of the computer)  
(for software, the priority is changeable, on line)
- its residence (hardware tasks are always resident, such as external subroutines).

RTL/2

- Priority.

13- Does a PA always have a name?

ALGOL 68

There is no mechanism for giving names to a PA - unless it is a procedure call and therefore could be construed to be a generation of that procedure.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

Yes, always.

PL/I

No, not required.

PROCOL

Yes, always.

RTL/2

A PA will be associated uniquely with a STACK and the identifier of the STACK could be considered to be the name of the PA.

14- How is a PA declared (implicitly/explicitly)?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Not relevant.

PEARL

Yes, explicitly.

PL/I

Either explicitly or contextually.

PROCOL

Yes, explicitly.

RTL/2

Implicitly.

15- Give naming sequence of a PA.

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Not relevant.

PEARL

See 5.

PL/I

Any allowable variable name.

PROCOL

- At generation level: Q/HARD TASK LIST ?  
A/SCRUTL, SCRUT2 ;  
.....  
Q/RESIDENT TASK ?  
A/RET1, RET2 ;  
.....  
Q/DESCRIPTION OF RES TASK  
A/RET1 = <priority>[, SEM (ij)] ;  
.....  
Q/DESCRIPTION OF NON RES.TASK  
A/NRET1 = <partition -nb>  
[,<priority>][, SEM (ij)] ;
- At language level: \* TASK  
[ Segment  
END;

RTL/2

In the STACK declaration.

16- How can data be passed between PAs?

ALGOL 68

By means of variables that are in scope to the PAs.

CORAL 66

Block sructure and parameters

GLOBAL

PAS 1 - PL/I

COMMON variables and parameter of procedure calls (block structure).

PEARL

As in ALGOL 68.

PL/I

By EXTERNAL variables, TRANSIENT files and data sets.

PROCOL

COMMON and files.

RTL/2

Via variables in databricks, files etc.

Relations between PCEs and PAs.

17- Can a PCE be distinguished from a PA?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Not relevant.

PEARL

Yes (see 5).

PL/I

Yes.

PROCOL

No.

RTL/2

Yes.



18- Is the association between a PCE and a PA static or dynamic?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I '

Not relevant.

PEARL

Both static and dynamic.

PL/I

Dynamic.

PROCOL

Not relevant.

RTL/2

Static or dynamic depending on level of system.

Static through system generation.

Dynamic through a procedure call (MAKETASK).

19- How is this association made?

Give an example.

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Not relevant.

PEARL

<u>Static</u>	<u>dynamic</u>
TASK T1 :	TASKNAME T2
BEGIN;	ACTIVATE T2:
.	CALL P;
.	(P is any procedure)
.	
END;	

PL/I

In the task activation statement.

PROCOL

Not relevant.

RTL/2

MAKETASK (stackname, procname, priority).

Priority of PAs

20- Does a PA have a priority?

ALGOL 68

No.

CORAL 66

Yes, at system generation time.

PAS 1 - PL/I

Yes.

PEARL

Yes.

PL/I

Yes.

PROCOL

Yes, at system generation time.

RTL/2

Yes.

21- How is priority defined or assigned?

ALGOL 68

Not relevant.

CORAL 66

At system generation time.

PAS 1 - PL/I

Within the call statement (PL/I),  
by the priority of the connected event in PAS 1.

PEARL

In the ACTIVATE and CONTINUE statement.

PL/I

In the task activation statement or by assignment to or initialization of the task variable.

PROCOL

At system generation time.

RTL/2

When the PA is created, through MAKETASK or at system generation.

22- Is priority absolute or relative to that of the PA that assigns it, or both?

Not relevant.	ALGOL 68
Absolute only.	CORAL 66
Absolute in PAS 1 relative in PL/I.	PAS 1 - PL/I
Absolute and relative.	PEARL
Relative only.	PL/I
Absolute only.	PROCOL
Absolute only.	RTL/2

23- Can a PA change its priority?

Not relevant.	ALGOL 68
No.	CORAL 66
No.	PAS 1 - PL/I

PEARL

Yes, by explicit statement: CONTINUE PRIORITY n.

PL/I

Yes.

PROCOL

No.

RTL/2

Yes.

24- Can a PA interrogate its priority?

ALGOL 68

Not relevant.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

No.

PL/I

Yes.

PROCOL

No.

RTL/2

Yes.



Resources of PAs.

25- What is the strategy of assigning resources to a PA?

ALGOL 68

Resources are allocated to each of the parallel activities that make up a parallel clause when the statements are executed. Explicit reservation is performed by semaphores.

CORAL 66

The strategy in CORAL is to allow a CORAL programmer to explicitly define the resource and dependencies in terms of the incline code inserts which 1) make him fully aware of what he is doing and 2) allow him to use the functions of the hardware of the machine where if it was a high level language function itself, it could be clumsy to implement on all machines, as the logic of these types of functions depends more on structured context with the particular functions available on the hardware architecture used.

PAS 1 - PL/I

Resources are assigned and reserved implicitly when the execution of a statement is performed. Explicit reservation of external standard devices by OPEN and LOCK statements ( PL/1). Explicit reservation by semaphores (PAS 1 and PL/I).

PEARL

No strategy for system-supplied resources ( they are handled by the O.S.). Resources created by the programmer can be handled by synchronizer variables (semaphore and bolt).

PL/I

Resource allocation is dependent on resource type. Processor time is basically assigned by priority. Storage can be assigned statically or dynamically. Peripheral devices can be assigned dynamically.

#### PROCOL

Simplicity in the first implementation, for relocation is not possible. Distinction is made between core and logical resources. Management of core is made by the O.S.. If the task has the highest priority at some time and is not in core, the scheduler asks for its introduction (needing eventually a swapping from backing store) and gives the control to ask in core having the highest priority.

A logical resource is either an external subroutine, a data area or a task. In case of a task the semaphore is attached to the task at S.G. time. The user has the possibility to delay the activation of a task T if another task T' is running until the completion of T'. In this case at S.G. time, he associates a semaphore to this task T.

#### RTL/2

The major resource is the STACK for the PA and this is assigned either at system generation or through MAKETASK.

Resources supplied by the O.S. (e.g. I/O) are assigned by the O.S. through procedure calls.

The user may use semaphores for the control of private resources.

26- When are resources allocated to a PA?

#### ALGOL 68

When the activity is started stack and heap space would be allocated.

#### CORAL 66

Reserved at S.G. time and connected on activation of a path through a PA.

#### PAS 1 - PL/I

At system generation time for PAS 1 tasks. In the actual implementation it is always done at S.G. time for PL/1.

PEARL

At task declaration time ( RESIDENT-attribute).

At task-activation time (priority option/ RESIDENT-attribute for task variables).

At task-execution time, when performing a bolt operation.

When entering a block containing declarations.

PL/I

At task initiation and dynamically.

PROCOL

At execution time, task waiting for the release of a particular resource are queued and hold it as soon as it is released.

RTL/2

The STACK may be allocated at system generation time or at time of call of MAKETASK; other resources at execution time.

27- Is it possible to guarantee a PA's resource requirement (e.g. a) core residence, b) processor time)?

ALGOL 68

- a) Not possible
- b) not possible.

CORAL 66

- a) Possible
- b) not possible.

PAS 1 - PL/I

- a) Not relevant
- b) not possible.

(\*)

PEARL

- a) Possible
- b) not possible.

PL/1

This question is outside the scope of the language. An implementation could be envisaged such that under most circumstances both storage and processor time requirements could be computed. It is easy then to see how a task's storage requirement could be generated but not its processor time. The computation of this last is a major exercise in systems engineering requiring much subjective judgement by an experienced systems engineer for any real time system of a reasonable level of complexity.

PROCOL

- a) Possible
- b) not possible.

RTL/2

- a) Possible
- b) not possible.

28- Can the status of resources be investigated and by what means?

ALGOL 68

No.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

No.

PL/I

Some, by various means.



PROCOL

No.

RTL/2

In some systems it might be possible to find out whether or not a particular STACK was assigned.

29- Can resources be allocated explicitly to a PA by the programmer? If 'yes' than which can be allocated (e.g. working space, processor etc.)?

ALGOL 68

Yes, semaphores can be used to ensure sole use of files and working variables by a particular PA. In addition each PA would have its own stack and heap.

CORAL 66

Yes, working space at S.G. time.

PAS 1 - PL/I

Yes, by use of semaphores external devices by use of OPEN/CLOSE statements.

PEARL

Yes, working space by RESIDENT option, data by bolt variables.

PL/I

Most resources need not be. Sufficient information can be specified to allow an implementation to explicitly allocate them.

PROCOL

Yes, RESIDENCE at S.G. time.

RTL/2

Working space ( the STACK) is explicitly assigned.  
Private resources by semaphores.



30- Give examples of such explicit resource-allocations.

ALGOL 68

```
sema s = level 1; int x;  
par (p1, p2, p3);
```

If the resources p1, p2 and p3 all use x then either procedure may ensure sole use of x during a code sequence by bracketing it between

```
down s; ... up s;
```

CORAL 66

```
SEMAPHORE ( );
```

PAS 1 - PL/I

REQUEST; RELEASE for semaphores

PEARL

For bolts: RESERVE, FREE, ENTER, LEAVE  
for semaphores: REQUEST, RELEASE.

PL/I

For storage, ALLOCATE and FREE.

PROCOL

REQUEST, RELEASE for SEM(i).

RTL/2

SECURE (), RELEASE () for semaphores.

### Happenings

31- What types of happening are distinguished in the language, and how is each defined?

ALGOL 68

The only type of happening that can give rise to a change in the running state of a parallel activity is the up or down operation on a semaphore.

CORAL 66

Not in language.

PAS 1 - PL/I

Free happenings in PAS 1 -programs.

Examples: E1 = INTERRUPT (13);

E2 EVERY 5 SEC;

E3 AT 12.30;

connected happenings in PL/I -programs: DCL E EVENT;

connected happenings in PAS 1 -programs: implicit declaration.

Loose happenings are not explicitly declared.

PEARL

There are: free happenings, loose happenings, connected happenings.

Examples: loose happening:

RESPONSE PRINTERERROR : GO TO L;

free happening:

ON INTERRUPT 1

ACTIVATE TA1;

connected happening:

RELEASE SORTED;

PL/I

PL/I has two types of events, synchronous and asynchronous. Synchronous events are called CONDITIONS. The programmer may code ON statements to process various CONDITIONS. If the CONDITION arises during execution of a task, execution of the current statement is suspended and control transferred to the relevant ON statement (or block). The programmer may use the SIGNAL statement to force such transfer.

Asynchronous events are controlled by EVENT variables. Such variables can be used to synchronize tasks or asynchronous events within a single task.

APG/7 has capabilities (external to PL/I) for association of tasks with time schedules, operator requests and process interrupts.

PROCOL

Connected happening:

normal end of a task: WAIT taskname  
COMPLETED (OR KILLED);  
starting a timer or a pulse counter.

free happening:

end of I/O: WAIT t  
symbolic interrupt: WAIT IT (nb);  
timer counter: WAIT  
tops number TIM (j)  
pulse counter: WAIT  
pulses-nb PULSE (j);  
(not allowed for hardware task)  
loose happening: EXIT of a task (Hard or Soft).

RTL/2

Events controlled by SET, WAIT, RESET.  
Semaphores controlled by SECURE, RELEASE.  
Loose happenings through standard error mechanisms.

32- Please give examples of all linkings of happenings with the execution of PCEs.

ALGOL 68

Parallel activity linkage is by semaphores. It is possible for a specified procedure to be called in some I/O error situations.

CORAL 66

Not relevant.

PAS 1 - PL/I

Free and connected happening in PAS 1-program:

SEQDO;  
ON EX ... END;  
SIMDO;

CORAL 66 (continued)

connected happening in PL/I-program:

WAIT (EX);

loose happening in PL/I-program:

EX:ON ENDPAGE BEGIN

.

.

.

END;

PEARL

See 31.

PL/I

Execution of an ON statement replaces, within that block, any previous ON statement for that condition. If no ON statement is provided, there is a system default provided. The programmer can REVERT to the previous ON statement or explicitly to the standard system action. The ON statement or block is executed under the control of the current task.

EVENT variables are under explicit programmer control unless associated with certain specific items.

PROCOL

See 31.

RTL/2

See 31.

33- Can a user enable/disable free happenings, at least to the extent of controlling interrupt connection to other PAs?

ALGOL 68

Not relevant as 'free happenings' are not recognized by the language.



CORAL 66

Not dynamic, only static.

PAS 1 - PL/I

Yes: CONNECT (n);  
DISCONNECT (n);

PEARL

Yes: ENABLE  
DISABLE

PL/I

APG/7 provides ENABLE, DISABLE and RESTORE statements which are, however, machine dependent.

PROCOL

Yes: MASK IT (i);  
UNMASK IT (j);  
SIGNAL IT (k);  
ACK IT (l) (this is referring to an "interrupt handling" session).

RTL/2

Not relevant.

Hierarchy of PAs

34- Can an PA hierarchy exist? If yes, how is it realized?

ALGOL 68

A PA hierarchy can exist and is determined by the block structure of the PCEs.

CORAL 66

In general, no.



PAS 1 - PL/I

The hierarchy for PL/I-tasks is defined by the block structure of the PCEs. There is no hierarchy for the PAS 1-tasks.

PEARL

Yes, the PA hierarchy is defined by the block structure of the PCEs.

PL/I

Yes. (\*)

PROCOL

No, a PA may initiate another PA as an independent entity. (\*)

RTL/2

Yes, but all subtasks are independent.

35- Is there the same mechanism for handling both independent and dependent PAS?

ALGOL 68

No mechanism for independent PAS exists. All activities started in one parallel clause must finish before the activity containing the parallel clause can finish.

CORAL 66

Same mechanism. (\*)

PAS 1 - PL/I

No. Independent PL/I tasks can only be started by the operator. Dependent PL/I tasks by activation of subtasks. PAS 1 tasks by completion of a connected event.

PEARL

No, dependent PA or subtasks must be terminated before the master-tasks can be finished.

PL/I

Yes, in APG/7, but there are no independent tasks known to PL/I. Communication between such may be effected by EXTERNAL variables (including EXTERNAL EVENT variables) as the language does not prohibit independent tasks.

PROCOL

There are only independent PAs.

RTL/2

All PAs independent.

36- When priorities exist, is there a dependency/relationship between the PA hierarchy and PA priorities?

ALGOL 68

Not relevant.

CORAL 66

Yes (\*)

PAS 1 - PL/I

Yes.

PEARL

Yes.

PL/I

Yes and no. That is, priorities are stated relative to the creating task but generate absolute values.

PROCOL

No.

RTL/2

No.

37- State any limitations on the number of PAs and hierarchical levels?

ALGOL 68

Depends on implementation.

CORAL 66

No limitation imposed by our hardware. (\*)

PAS 1 - PL/I

Depends on implementation.

PEARL

Depends on implementation.

PL/I

There is no implicit limit in the language.

PROCOL

63 maximum tasks (hardware or software tasks).

RTL/2

Depends on implementation.

PA commands and operations

38- Are PCE/PA operations included in the language? - How?

ALGOL 68

Only the semaphore operators up and down are available.

CORAL 66

Yes, by procedure calls.

PAS 1 - PL/I

Yes, by statements and standard built in functions.

PEARL

Yes, by statements.

PL/I

All task and procedure control facilities are available in the language.

PROCOL

Yes, by a set of statements.

RTL/2

By procedure and SVC calls.

39- Can a dependent PA issue all PA operations?

ALGOL 68

Yes.

CORAL 66

Yes. (\*)

PAS 1 - PL/I

Yes.

PEARL

Yes.

PL/I

Yes.

PROCCL

Not relevant. There are no dependent PAs.

RTL/2

Yes.

40- What commands can be executed by a PA operating on another PA (not itself)? Please give examples for each.

ALGOL 68

A PA may operate on another PA by an up or down on an associated semaphore.

CORAL 66

Not relevant.

PAS 1 - PL/I

PL/I-PAS: start of PAS1-PA by COMPLETION (E)='1'B;  
start of PL/I-PA by CALL P PRIORITY (...). . . ;  
continuation of PL/I-PA by COMPLETION (E)='1'B; (PA may wait for event E)  
PAS1-PAS: start of PAS1-PAS continuation of PL/I-PAS by SET (E);  
(E is an event, connected to a PAS1-task and/or a PL/I-task is waiting for)

PEARL

ACTIVATE - statements  
SUSPEND - statements  
CONTINUE - statements  
DELAY - statements  
TERMINATE - statements  
PREVENT - statements

PL/I

One task may activate another task, change or detect its relative priority, continue all tasks waiting on an event and when terminating, will terminate all dependent tasks.

PROCOL

ACTIVATE <hard- or soft task>;  
ACTIVATE <soft task>  
WHEN <i-o unit - nb>;



PROCOL (continued)

```
ACTIVATE <soft task>
WHEN <indust. unit>;
EVERY <nb -top> TIM  (<timer - nb>)
ACTIVATE <soft task>;
AFTER <nb - top> TIM  (<timer - nb> )
ACTIVATE <soft task>;
EVERY <nb - top> PULSE (<pulse - nb> )
ACTIVATE <soft task>;
AFTER <nb - top> PULSE (<pulse - nb> )
ACTIVATE <soft task>;
CANCEL soft task ;
```

RTL/2

```
MAKETASK (stack, proc, priority)
START (stack)
STOP ( stack)
KILL (stack)
```

41- What commands can be executed by a PA operating on itself?  
Please give examples for each.

ALGOL 68

The PA can terminate or halt itself.

CORAL 66

Not relevant.

PAS 1 - PL/I

By STOP; the task terminate itself and dependent tasks.

PEARL

```
SUSPEND
CONTINUE
DELAY
TERMINATE
PREVENT
```

PL/I

A task may wait on an event, delay for a time period, terminate and change its priority.

PROCOL

ACTIVATE itself if soft task ; and other forms of ACTIVATE.

EXIT; last instruction which must end any task.

WAIT (all forms of wait generated by a software task).

RTL/2

DELAY (time)

STOP (stack)

WAIT (event)

SECURE (sema)

42- Is there a difference in effect when the same PA command is given either to a dependent or an independent PA?

ALGOL 68

No.

CORAL 66

Would depend on implementation.

PAS 1 - PL/I

No.

PEARL

No.

PL/I

Independent tasks have no formal knowledge of one another but the language does not preclude their existence and allows communication between them through EXTERNAL variables and TRANSIENT files.

PROCOL

(\*)

RTL/2

No dependent PAs.

Timing facilities

43- Is there a timing mechanism?

ALGOL 68

No.

CORAL 66

No.

PAS 1 - PL/I

Yes.

PEARL

Yes.

PL/I

Yes.

PROCOL

Yes.

RTL/2

Yes.

44- What can the timing mechanism schedule?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Tasks (\*)

PEARL

Execution of task-operations at specified points of time and/or execution of task-operations after specified time intervals.

PL/I

PL/I allows time delays in a task. APG/7 allows task scheduling on time of day or time intervals to be specified external to the language.

PROCOL

Software tasks only can be scheduled.

RTL/2

Delays to tasks.

45- What elements exist for timing?

ALGOL 68

Not relevant.

CORAL 66

Not relevant.

PAS 1 - PL/I

Clock constants and duration constants.

PEARL

In PEARL there are new data types for absolute time (CLOCK) and for time-intervals (DURATION). It is possible to express clock- and duration constants and to declare clock- and duration variables.

```
(DECLARE S DURATION:= 2HPS;  
DECLARE T CLOCK:= 12:20:5;)
```

PL/I

DATE built in function and DELAY statement. APG/7 also has TIMER and PULSE attributes.

PROCOL

Symbolic 'timers' and 'pulse' counters with 'tops'

START timer ; STOP timer ;

START pulse-counter ;

STOP pulse-counter

There exist several different timers and pulse-counters.

RTL/2

None.

46- Is it possible to specify time(s) and/or time interval(s) as an option of a PA command, e.g. by AFTER, AT, ALL, DURING, UNTIL options.

Please specify the commands for which this is possible.

ALGOL 68

No.

CORAL 66

No.



PAS 1 - PL/I

Yes, in the declaration of PAS1 events: AFTER

WAIT

AT

PEARL

Yes, for all PA operations.

PL/I

No.

PROCOL

Yes, AFTER, WAIT, EVERY for symbolic times and pulse counters.

RTL/2

Yes, for SECURE and WAIT.

47- What happens after scheduling a PA ( to the PA and/or other PAs)?

ALGOL 68

Usual system operation or a queue (O.S. dependent).

CORAL 66

Usual system operation or a queue (O.S. dependent).

PAS 1 - PL/I

Usual system operation or a queue (O.S. dependent).

PEARL

Usual system operation or a queue (O.S. dependent).

PL/I

Implementation dependency.

PROCOL

Usual system operation or a queue (O.S. dependent).

RTL/2

Usual system operation or a queue (O.S. dependent).

48- Is there a relation between scheduling and the reaction on happenings? Please give the mechanism(s) by which this is implemented for free and loose happenings.

ALGOL 68

This question is outside the scope of the language report.

CORAL 66

Would depend on implementation.

PAS 1 - PL/I

Free happenings are handled by PAS1:

```
eventname = INTERRUPT (number of interrupt);
```

.

.

.

```
                SEQDO;  
ON eventname    taskbody END;  
                SIMDO;
```

Loose happenings may be handled in PAS1:

```
                SEQDO  
ON happening name    taskbody END;  
                SIMDO
```

or in PL/I:

```
ON happening name GOTO label.
```

PEARL

Yes.

Example for free happenings: ON interrupt ACTIVATE task;

Example for loose happenings: RESPONSE signal GOTO label.

PL/I

External to language consideration.

PROCOL

(\*)

RTL/2

(\*)

Synchronization of PAs

49- What facilities exist for explicit synchronization between PAs, or between PAs and I/O devices in the language?

ALGOL 68

Semaphores.

CORAL 66

Not relevant.

PAS 1 - PL/I

Semaphores, events, and OPEN/CLOSE operations.

PEARL

Semaphores, bolts.

PL/I

EVENT variables, ON conditions and WAIT statements.

PROCOL

Primitives on semaphore, symbolic interrupt, I/O and for task:  
ACTIVATE soft task WHEN i/o unit - nb or indust. unit ;

RTL/2

Semaphores, events.

50- Are PAs also implicitly synchronized? If so, how?

ALGOL 68

No.

CORAL 66

Not relevant.

PAS 1 - PL/I

Only in the case of sharing external devices. Termination of a task terminates all its subtasks regardless of their status.

PEARL

A maintask waits at the end of a block, as long as subtasks contained in this block are not terminated.

PL/I

No.

PROCOL

Different incarnations of a program as a task are implicitly synchronized.

RTL/2

No.

51- Is the acces of the PAs to different kinds (or classes) of resources synchronized in only one way? If not, explain the different synchronization mechanisms as far as possible.

ALGOL 68

No. (\*)



CORAL 66

Would depend on implementation.

PAS 1 - PL/I

The access to I/O devices is synchronized by the O.S.  
For other synchronization problems "semaphores" and "events"  
may be used.

PEARL

The access to I/O devices is implicitly synchronized by the O.S.  
For mutual exclusion problems the BOLT-variables may be used.  
Producer-consumer problems may be solved by use of SEMAPHORES.

PL/I

Yes, certain types of data sets may be locked at the record level. ON conditions may be inhibited across multiple statement groups. Messages may be passed via TRANSIENT files without explicit synchronization and EVENT variables can be used to control access to resources.

PROCOL

(\*)

RTL/2

No.

52- How can the status of a PA be investigated by means of language (by another PA)?

ALGOL 68

The integer value of a semaphore can be investigated using the level operator.

CORAL 66

Not relevant.



PAS 1 - PL/I

Normal termination by checking the completion value of the event.

PEARL

Not possible.

PL/I

Only the priority of another task may be investigated.

PROCOL

Not possible at this time.

RTL/2

Not possible.

53- For each explicit synchronization mechanism in the language, is there a TIMEOUT/ELSE option? Give example of use.

ALGOL 68

No.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

No.

PL/I

No.

PROCOL

No.

RTL/2

Yes. TWAIT (event, interval, faillabel) etc.

-61-

INPUT/OUTPUT

PRECEDING PAGE BLANK-NOT FILMED

General method of I/O handling in the language

1- Which I/O-facilities are there on the language level?

If none, describe briefly how I/O is done otherwise (e.g. code procedures, macro-features).

Please give some examples.

ALGOL 68

ALGOL 68 performs all I/O using procedure calls. There are also the associated modes file and format.

Examples of I/O procedure calls are:

```
open (x, "dsn=h.p, disp=shr" 2);
establish (x, "dsn=h.p, disp=shr", 10, 5, 80, 2);
print (r);
put (x,r);
```

CAMAC - IML

- Declaration of access routes to devices (singly or in groups).
- Declaration of areas in computer memory used for I/O.
- Declaration of interrupt sources.
- Transfer of one word to or from one device or single control action.
- Transfer of a sequence of words to or from one device, or repetitive control action on device.
- Transfer of a sequence of words to or from a sequence of devices, or control action on all devices in sequence.
- Action on interrupt sources.
- Action on the access routes to devices.

CORAL 66

None. Normally achieved by Macros which insert code statements or via library procedures.

CORAL 66 (Continued)

Example:

To copy a string of characters packed three to a word

```
'DEFINE' PUNCH (Q) " 'CODE' 'BEGIN'
                                SETA.Q;
                                SETB.5.1;
                                LINK.58
                                'END' " ; (MACRO DEFINITION OF PUNCH)
'DEFINE' FALSE "0" ; (MACRO DEFINITION)
'DEFINE' TRUE " 1 " ; (MACRO DEFINITION)
'PROCEDURE' COPY ('VALUE' 'INTEGER' STRING);
'BEGIN'
    'INTEGER' TERMINATOR, CHAR, WORD;
    TERMINATOR:=FALSE;
    'FOR' STRING:= + 1 'WHILE' TERMINATOR = FALSE 'DO'
    'BEGIN'
        WORD:= [STRING-1] ;
        'FOR' CHAR:= 'BITS' [8,16] WORD, 'BITS' [8,8] WORD, 'BITS' [8,0]
                                                WORD 'DO'
    'BEGIN'
        'IF' TERMINATOR = FALSE
        'THEN'
            'BEGIN'
                'IF' CHAR =61
                'THEN' TERMINATOR:= TRUE
                'ELSE' PUNCH (CHAR)
            'END'
        'END'
    'END'
'END'
```



#### PAS 1 - PL/I

All I/O instructions are on language level.

Formatted, unformatted, organized and not organized I/O from and to standard peripherals.

Primitive, checked and calibrated process-I/O.

No graphic I/O.

Process-I/O example: IN PRESS; OUT SWITCH (I);

#### PEARL

The input-output facilities in PEARL comprise the flow of data (communication) and the organization of data storage and retrieval (file handling).

#### PL/I

The language offers input and output support in two basically different forms, STREAM and RECORD, and for all data types.

#### PROCOL

Two kinds of I/O facilities exist: classical-I/O is distinguished from process-I/O.

Classical-I/O statements are the usual READ and WRITE statements for the classical side in the FORTRAN point of view.

Process-I/O have been designed similarly as the former, using formats and symbolic naming of the peripherals involved.

#### RTL/2

The RTL/2 language as such contains no specific provision for input and output since to do so might prove an undesirable burden for some systems. However in order to aid transportability of programs between systems a recommended standard package for character streaming has been defined. This is briefly outlined

## RTL/2 (Continued)

below and described in detail in the manuals 'Standards for RTL/2 systems' and 'Standard stream I/O facilities within RTL/2 systems' which should be consulted for details of formats, errors etc.

### Streaming mechanism

Each task has two SVC data bricks associated with it, namely:

DATA RRSIO;		DATA RRSED;
PROC (BYTE) IN;		BYTE TERMCH
	and	BYTE IOFLAG;
PROC (BYTE) OUT;		
ENDDATA;		ENDDATA;

The procedure in IN will remove the next character from the current input stream and return it as result. The procedure in OUT will send the character passed as parameter to the current output stream. All streaming of individual characters will be via IN and OUT as appropriate.

TERMCH and IOFLAG are concerned with the standard stream input procedures.

### Input

Individual characters are obtained from the current input stream by calls of the procedure variable IN in data brick RRSIO.

Numbers and text may be read from the current input stream by the following procedures. In each case the last character read and removed (the terminating character) is placed in TERMCH in data brick RRSED.

PROC FREAD ( ) FRAC reads a signed decimal number and returns a truncated fraction value as result.

PROC IREAD ( ) INT reads a signed decimal integer and returns its value as result.

PROC RREAD ( ) REAL reads a signed decimal number with optional exponent and returns its value as result.

RTL/2 (Continued)

PROC TREAD (REF ARRAY BYTE X,T) INT reads characters and places them into successive elements of X. Input is terminated as soon as one of the characters of T is encountered. The number of characters placed in X is returned as result.

Output

Individual characters are sent to the current output stream by calls of the procedure variable OUT in data brick RRSIO.

Numbers and text may be output to the current output stream by the following procedures.

PROC NLS (INT N) and PROC SPS (INT N) send N newline and space characters respectively.

PROC FWRT (FRAC X) sends the unrounded fraction value X as signed decimal number in a fixed format dependent upon the implementation.

PROC IWRT (INT X) sends the integer value X as a signed decimal integer with leading zeros suppressed.

PROC RWRT (REAL X) sends the unrounded real value X as a decimal number in a fixed format dependent on the implementation.

PROC FWRTF (FRAC X, INT N), PROC IWRTF (INT X,M) and PROC RWRTF (REAL X, INT M,N) send the fraction, integer and real values (rounded where appropriate) in formats determined from the values of the additional parameters M and N.

PROC TWRT (REF ARRAY BYTE A) sends the successive elements of the array A as characters.

2- Please describe in short form the scope of the set of I/O facilities in the language.

(As guidelines:

"only simple mechanisms: I/O only of single data items, only binary transfer, only single hardware registers addressable" etc.

or:

"a comprehensive system with character-, graphic-, process-I/O, different formats" etc....)

#### ALGOL 68

Character and binary I/O, with elaborate formatting facilities in the former. Random access is possible if the channel (i.e. type of file) permits.

#### CAMAC - IML

IML is not a complete language, but allows detailed specification of I/O actions in CAMAC systems (see glossary). It contains full provision for defining access routes (datapaths) between the computer and the CAMAC devices, and for defining the manner and sequencing of data transfers. Data are transferred directly between variables, arrays or buffers in computer memory and registers or pseudo-registers in CAMAC devices, without alteration (formatting or scaling, etc.). It relies on a host environment to provide facilities for data processing and communication other than with CAMAC.

As a consequence, a number of the questions are not applicable to IML, dealing with more high-level facilities. Such high-level facilities could be obtained by programming in host language and IML.

#### CORAL 66

The I/O facilities being based on the Code Statements normally called up by Macros or system procedures are capable of making the optimum use of any hardware facility on any particular machine. Hence, by a properly constructed set of system Macros or Procedures, The I/O facilities can be made to be as comprehensive as desired.

#### PAS 1 - PL/I

A practicable set of I/O-instructions for all kind of I/O is contained.

stream I/O : GET/PUT

record I/O : READ/WRITE

formatted I/O: EDIT



#### PAS 1 - PL/I (Continued)

standard formatted I/O: LIST

unformatted I/O: READ/WRITE

process I/O: IN/OUT.

#### PEARL

The data sources and data sinks which communicate by the flow of data in a process control system may be physical devices (e.g. disk memory, ADC) or logical devices (e.g. files, working storage).

The programmer can address a physical device if its name occurs in the system-division. Logical devices require additional organization. In the case of files storage areas are established by file-handling-statements. In the case of working storage the compiler is responsible for this organization.

PEARL provides a complete set of conventional I/O together with tools for graphic data handling and inter-system data transfer.

#### PL/I

The language allows a rich variety of data organizations on the source and sink devices and allows unlimited data transfer with a single statement.

#### PROCOL

Character, binary and analog I/O with elaborate formatting facilities specially suitable for Process Control applications. Formats allow handling of Physical Control of an input, filter, logcheck, etc.... tailored to the need of the user.

- 3- Is the system self-contained or does the user have the possibility to create new elements for new kinds of I/O and/or new devices?

(Extensible system; c.f. I/O-requirements, point 3.2 and point 4 e.g. which mechanism on source-language level is there which allows the inclusion of sequences of I/O-commands on machine-language level?



3-(Continued)

or:

Can new mechanisms be described by combinations of already existing language elements?)

ALGOL 68

The system is self-contained and one can only I/O to devices that are legitimate channels. The association of the channel with the I/O device is a JCL problem and is outside the scope of the language.

CAMAC - IML

The CAMAC part is self-contained. Any new elements introduced (i.e. new modules) will use standard CAMAC function codes, and can therefore be used without amendment to the language.

CORAL 66

Since CORAL 66 is designed to allow code inserts wherever a CORAL 66 statement may be made, all available hardware facilities may be exploited. (In this context 'code' may be the machine code, assembler code, or any higher level - depending on the particular implementation). Hence, the system is fully extensible.

PAS 1 - PL/I

The system is self-contained and there are no extensions possible on language level.

PEARL

In principle self-contained but by use of primitive I/O (MOVE) procedures for handling non-standard devices can be programmed.

PL/I

The language can be extended by definition of language sequences to a preprocessor and by creation of callable subroutines in assembly language.

#### PROCOL

The system is self-contained; it is not restricted to one kind of device but may be extended to take into account new kind of devices (the worst way to achieve this would be by using code block inserts).

In order to have the greatest flexibility, the programs can define the peripherals on language level in a symbolic manner.

4- What is the overall structure of the I/O-mechanism?

For explanation:

Is there a strictly modular structure, i.e. is a complete I/O-instruction a combination of principally independent elements. (e.g.: Modules for initiation of devices, formatting, parameter passing)

or is there a largely complete and self-contained set of statements of possibly similar construction for the different purposes (e.g: independent mechanisms for character-, graphic-, process-I/O)?

#### ALGOL 68

There is a set for procedure calls for open, read, write, close, etc.

#### CAMAC - IML

I/O actions can be classified as data transfers and control actions. Control actions are basically similar to data transfers except that no data are involved. The particular CAMAC function determines whether and how data are used. Data are transferred in units of up to 24 bits wide, depending on the particular hardware module concerned. As well as transfers in and out on addressed registers, there are explicit accesses to associated status and control information at module, crate, branch and system level. Each action uses one CAMAC function, and can be:

#### CAMAC - IML (continued)

- Single : between an individual register within a module and one word in computer memory.
- Block : between an individual register within a module (accessed at successive times) and a sequence of words in computer memory.
- Multiple: between a number of registers within modules (addressed either as a sequence or individually) and the same number of words in computer memory.

In both latter cases, the action may be carried out using an autonomous channel or by means of a series of single actions under software control. The timing control can be either by the computer program directly (sensing the module) or in response to demands (using the computer's interrupt system); termination may be either by the computer program directly (exhausting a count), or as a consequence of the Q-response from the module. (See also 33).

#### CORAL 66

The structure of the I/O mechanism is based on the use of machine code, or assembler, normally available to the programmer by means of system defined Macros or Procedures. Therefore the normal structure can be geared to suit the particular class of applications handled by any particular system implementation.

#### PAS 1 - PL/I

Instructions for standard and process-I/O are combinations of partly optional elements.

#### PEARL

There is a largely complete and self-contained set of statements of similar constructions for the different purposes (e.g.: independent mechanisms for character-, graphic-, process-I/O).

#### PL/I

The input/output statements are largely device independent although some special attributes are used to indicate the peculiarity of some devices.

#### PROCOL

1) Classical I/O:

A set of READ, WRITE and FORMAT instructions.

2) Process I/O:

- peripheral I/O declarations ( made in the COMMON unit)
- Process-I/O format definition (made in the COMMON unit)  
format for input are distinguished from format for output
- Process-I/O instructions: INPUT, OUTPUT.

The Process-I/O case has been handled in a way very similar to the classical part.

- 5- Does the language reflect some specific input/output hardware or allow the description of specific input/output hardware?

#### ALGOL 68

The language does not reflect any specific input/output hardware and furthermore such hardware cannot be described in the language.

#### CAMAC - IML

Specific input/output hardware, CAMAC as defined in EUR 4100, also (in some cases) EUR 4600.

Examples:

The hardware addressing, LAM handling, Q-response, and available functions for I/O and status information.

#### CORAL 66

Depends on system implemented.

#### PAS 1 - PL/I

The language does not reflect any specific I/O hardware. It allows to address all devices of the BBC-process-peripherie and a set of standard peripherals.



AD-A032 571

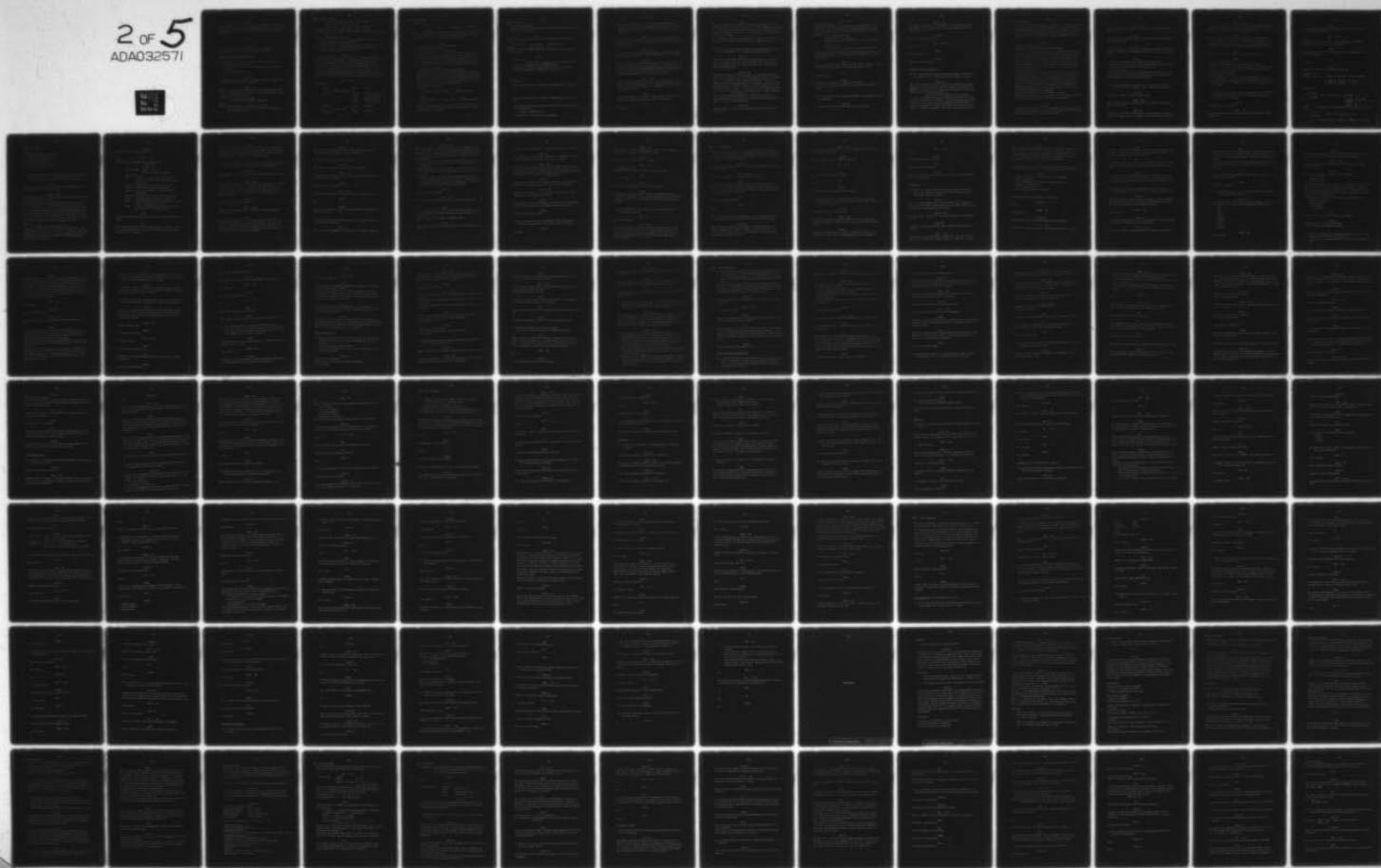
PURDUE UNIV LAFAYETTE IND PURDUE LAB FOR APPLIED IND--ETC F/G 9/2  
A LANGUAGE COMPARISON. A COMPARISON OF THE PROPERTIES OF THE PR--ETC(U)  
OCT 76 R ROESSLER, K SCHENK

N00014-76-C-0732

UNCLASSIFIED

NL

2 of 5  
ADA032571





PEARL

The description of specific input/output hardware is possible in a respective implementation, although there may exist some special hardware features which cannot be reflected by language elements.

PL/I

(\*)

PROCOL

No, only some classes of peripherals are distinguished: conventional, analog, digital, special (or default).

Syntactic form of I/O-instructions

- 6- Please describe briefly the general syntactic form of I/O instructions in the language
- in the case of statements on language level using special key-words
  - in case of procedure calls.

ALGOL 68

A standard ALGOL 68 procedure call with a number of parameters dependent on the particular procedure.

CAMAC - IML

Note IML semantics are fundamental. Different syntactic embodiments of them are being formulated. The following replies relate to the Macro Expansion Syntax.

A typical I/O instruction specifies

⟨stype⟩ ⟨action⟩ ⟨devadd⟩ ⟨memadd⟩

where

⟨stype⟩ denotes the sort of action e.g. SA, UBL

(SA = single action; UBL = Uni-device Block transfer synchronized by LAM)

# CAMAC - IML (continued)

<action> denotes the CAMAC function e.g. READ, BISET

(READ = F(0), transfer data in;

BISET = F(18), Selective set group 1 register)

<devadd> denotes the symbolic device address (See 13)

<memadd> denotes the symbolic memory address (if there is a transfer).

Other I/O instructions include

a jump-to label (e.g. if the CAMAC response Q is to be tested)

a repeat count (for compound actions)

a demand name (for a LAM-synchronized block transfer)

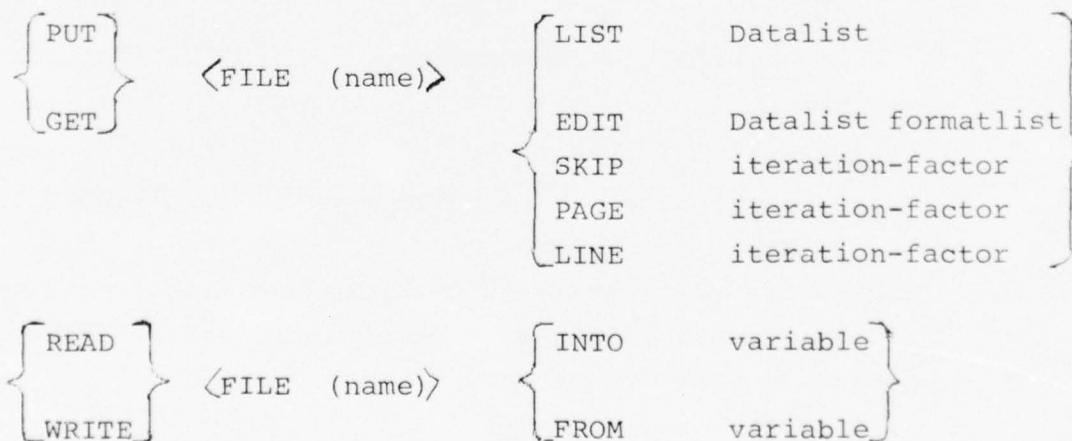
a channel number (for the data path and transmission mechanism to be used in a compound action).

## CORAL 66

The syntactic form for I/O is undefined in the language. Instead it is based upon the hardware facilities available, together with the requirements of the particular system. Some work is in progress to standardize on the names, parameters, etc... of a set of procedure calls or macros for I/O. Initially this is only for utility programs dealing with character and block transfers and file handling. Future work will consider the requirements of process I/O control and could conform to a LTPL synthesis.

## PAS 1 - PL/I

Standard I/O:



PAS 1 - PL/I (Continued)

Process I/O:

$\left. \begin{array}{c} \text{IN} \\ \text{OUT} \end{array} \right\}$       name      <gaugeproc>      <checkproc>

Name is the programmer defined name of a peripheral device, gaugeproc and checkproc are the names of programmerwritten procedures for gauging and checking transferred process data.

PEARL

There are four classes of I/C-instructions:

- not formatted and not organized

MOVE source TO sink      gauge-option

Primarily the MOVE-statement can only transport data in binary form without any transformation. The gauge-option allows a transformation of these values before output or after input. So this statement can be used for basic control functions of non-standard devices as well as for calibrated process-I/O.

- not formatted and organized

TRANSFER source TO sink [QUANT data-length]

Data transfer to and from files is performed by the TRANSFER-statement. The amount of data to be transferred is given either implicitly by the size of source or sink or explicitly by data-length in the QUANT-option. By default a transfer is finished if some end-of-file condition is encountered.

- formatted and not organized

$\left. \begin{array}{l} \text{READ} \\ \text{SEE} \end{array} \right\}$       sink      [FROM source]      [FORMAT format]

$\left. \begin{array}{l} \text{WRITE} \\ \text{DRAW} \end{array} \right\}$       [source]      [TO sink]      [FORMAT format]

Source and destination need not be explicitly mentioned in an I/O-statement. If there is no explicit reference, a standard source or sink is assumed.

PEARL (continued)

- formatted and organized

Same as formatted and not organized, but source or destination are files instead of devices.

PL/I

The general syntactic form of the I/O statement in the language is:

key-word-verb      FILE (name)      modifier-list;

where:              key-word-verb      is

READ/WRITE/GET/PUT/OPEN/CLOSE/SCAN and modifier list is key-word-verb dependent.

PROCOL

1) Classical I/O:

READ WRITE (peripheral-nb [format-nb] I/O-list

- Formats are compatible with FORTRAN formats
- There are instructions without formats (binary transfer)
- REWIND and BACKSPACE (for MITRA 15 version only).

2) Process I/O

INPUT ( industrial-unit, iformat-nb, complementary) variables-list

or

OUTPUT (industrial-unit, oformat-nb, complementary) variables-list

- complementary concerns the gain and/or filter characteristics of the peripheral

e.g. RANGE 10; FILTER TAB (1)

- iformat-nb or oformat-nb may be variable.



- 7- Does every instruction contain the complete description of the I/O-operation or is it possible to perform e.g. code transformations and formatting separately from the data transfer proper?

ALGOL 68

Every instruction (procedure call) contains a complete description of the I/O-operation and formatting cannot be separated from the data transfer.

CAMAC - IML

Each instruction describes the I/O-operation completely. There are no language elements for code transformation or formatting.

CORAL 66

Every statement can contain the complete description of the I/O-operation. Alternatively, it is possible to perform code transformations and formatting **separately** from the data transfer proper. It all depends upon the hardware facilities, code statements and system requirements of the given implementation.

PAS 1 - PL/I

Every instruction contains the full I/O-instruction, only the format list may be represented by a label name and the gauging and checking of process data may be separately from the I/O-instruction (see 6).

PEARL

With the exception of the MOVE-statement every instruction contains the complete description of the I/O-operation.

PL/I

No and yes.



#### PROCOL

Every I/O-instruction contains in the associated format the complete description of the I/O operation. In the T 2000-version, there is a possibility to use standard or particular conversion mechanisms. In the MITRA 15 version a dynamic choice of the mechanism to use next is possible.

In the MITRA 15 version, formatting can be made separately from the proper data transfer by the instruction: EXECUTE format-nb.

- 8- How are the formats and/or external devices identified within I/O instructions (e.g. numbers or symbolic names)?

#### ALGOL 68

Formats are associated with data items within the I/O instruction ( procedures inf, out, etc.). External devices are identified with files in the open procedure using the identification string and the channel.

#### CAMAC - IML

There are no formats in the language - all transfers are of binary words. An external device is identified solely by the access route to it: Branch, Crate, Station, Subaddress. This is its CAMAC address. No other properties are relevant: everything else is determined by the CAMAC standard. A device declaration introduces the symbolic name for the device at a stated CAMAC address, or the devices at a given (or calculated) set of CAMAC addresses.

LOCD SCALER, H, 1, 1, 12, Ø

is a local device declaration (i.e. valid in this program set only) for the identifier SCALER, at the Hardware address branch 1, crate 1, station 12, subaddress Ø.

#### CORAL 66

Numbers, symbolic names, or a mixture of the two descriptions may be used.

#### PAS 1 - PL/I

Formats: see 6 and 7.

Standard I/O devices are represented by file-names in the I/O instruction. The connection of file-names and devices is done in the declaration of the file and may be changed at run-time.

Example:

DCL XY STREAM INPUT CAR;

CAR is key-word for cardreader

Changing at run-time by writing on the console type-writer:

FILE (XY, TAR);

Now the (paper-) tape reader is connected with XY.

Process I/O devices are named and described within the so-called EQUIPMENT-description and these names are addressed in the I/O instruction.

#### PEARL

Formats and devices may be identified by symbolic names (or better identifiers) as each other programmer defined item.

#### PL/I

Symbolic names.

#### PROCOL

Formats are identified by numbers or variables.

Classical and Process I/O devices are identified by symbolic names.

9- Is there any formal distinction between I/O commands and status commands?

#### ALGOL 68

This question is outside the scope of the language.

CAMAC - IML

The CAMAC system treats all register in the same manner. There are, however, special functions for group-2-registers (status registers).

CORAL 66

Not in any systems we use.

PAS 1 - PL/I

No (see 14).

PEARL

See question 14.

PROCOL

Outside language scope.

Access to external devices

10- Are there different ways of handling classes of devices in the language (e.g. formatted, graphic, binary, process)?

ALGOL 68

The language intends different devices to have different mode channels on which different types of transput are possible. A channel and a file are associated with some data on an external device by call of procedure open, e.g.

open (file, identification string, channel).

In a given implementation certain channels may permit character transput, others binary transput, others process transput, etc... According to the report every implementation should permit sequential character I/O (channels standin and standout) and binary I/O ( channel standback). The former channels must permit

### ALGOL 68 (Continued)

both formatted and unformatted transput, but on the standback channel these concepts do not apply. Transput is performed by procedures which differ in the formatted, unformatted and binary cases. If more devices were added which cannot be accessed in this simple way then further channels and transput procedures would be needed.

### CAMAC - IML

The official CAMAC definition is:

#### Use of Q for Block Transfers: Address Scan Mode

If a module contains registers that are intended to be accessed sequentially in Address Scan mode they must be located at consecutive sub-address starting at  $A(\emptyset)$ . During read and write operations the module must generate  $Q=1$  at all sub-addresses at which these registers are present and  $Q=\emptyset$  at the first unoccupied sub-address, if any. A module with  $n$  such registers must therefore generate  $Q=1$  at  $A(\emptyset)$  to  $A(n-1)$ . If  $n < 16$  the module must generate  $Q=\emptyset$  at  $A(n)$ .

The Address Scan mode of block transfer is used for data transfers to or from an array of modules that do not necessarily occupy consecutive stations or all sub-addresses. The state of  $Q$  during each operation is used by the controller to determine the station number and sub-address for the next operation. When  $Q=1$ , the sub-address is incremented with carry-over into the station number. When  $Q=\emptyset$ , the sub-address is set to  $A(\emptyset)$  and the station number is incremented.

This allows unoccupied stations within an array.

The block transfer may be terminated by the controller on reaching a specified word count (recommended) or address.

### CORAL 66

As mentioned in question 1, I/O is handled via library type procedures or code inserts. As these can be tailored to any requirements and devices any device can be handled.



PAS 1 - PL/I

There is a difference between standard and process peripheral devices: see 8.

PEARL

All devices are treated as sinks or sources of data principally in the same manner.

PL/I

Yes, a distinction is made between STREAM and RECORD I/O devices, between TRANSIENT and non-TRANSIENT RECORD I/O devices and ANALOG devices.

PROCOL

All kinds of peripherals are handled in the same way:

- Definition of the symbolic unit corresponding to the peripheral
- Description of the format (for formatted I/O)
- I/O instructions.

In the case of Process Control peripherals the formats are somewhat extended to take into consideration specific problems of the transmission of non numeric information.

11- Are devices described implicitly in the I/O- instruction by key-words, device numbers, etc... and how is this done?

ALGOL 68

By channel numbers in the open procedure.

CAMAC - IML

There are no implicitly defined devices or modules. For explicit definitions see 8.

CORAL 66

Depends on system but at RRE we tend to open channels by specifying the channel name as a number or its actual name in a string.



PAS 1 - PL/I

Devices are addressed in I/O-instructions by programmer-defined names. The connection of the names with the devices and the description of the latter is done elsewhere (see 8).

PEARL

Source and destination need not be explicitly be mentioned in an I/O-statement. By omission of a respective reference a standard device is assumed. Any device or file can be declared as a standard sink or source by the standard-terminal-declaration.

PL/I

The file-name is used to designate the required device.

PROCOL

1) In the T 2000 Version:

Classical devices are named implicitly by numbers.

Part of the entire of a Process peripheral is named (declaration) in a symbolic way. The device is named by a number (system generation part).

2) In the MITRA 15 Version:

The definitions of the devices are made through the declaration: 'DEFINE' conformally with the MMT systems. This declaration is made in the COMMON. In this case the symbolic name is a number.

12- Are devices described by lists or information of some other kind which is generated by some "job-control-language" at link-time or at load-time?

ALGOL 68

Not specified in language.

CAMAC - IML

Device properties are not described, being determined by CAMAC standards.

# CORAL 66

Not on the RRE computers, some CORAL 66 system fix I/O paths at system generation time.

## PAS 1 - PL/I

No JCL.

Process devices are described in the systempart ("EQUIPMENT") of PAS 1-programs by lists of information.

## PEARL

Devices are made known to the compiler by the system-division of a PEARL-program.

## PL/I

Frequently.

## PROCOL

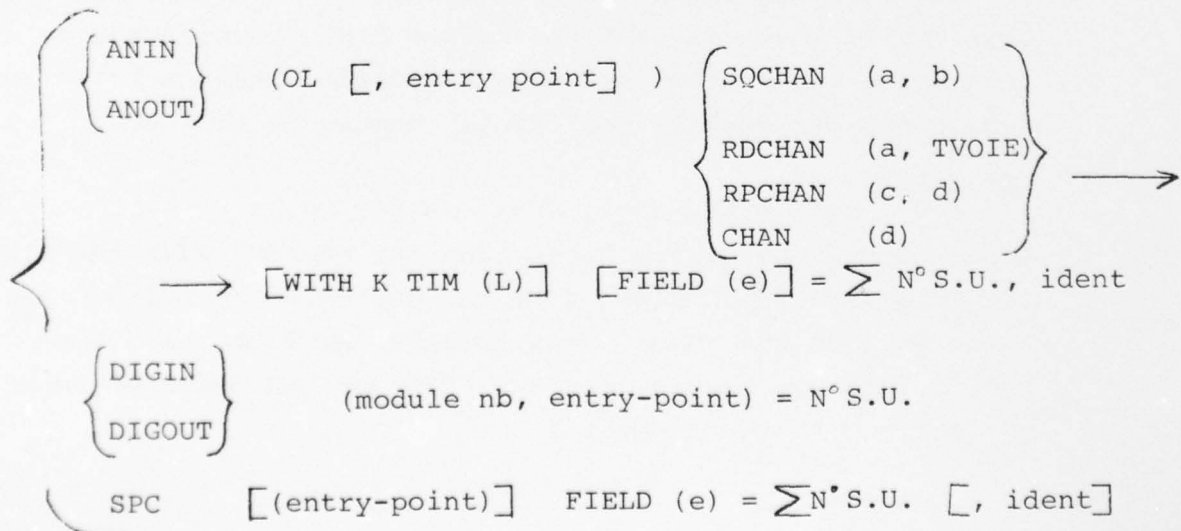
Devices are defined by the DEFINE instruction.

Classical I/O:

DEFINE  $n_1, n_2, \dots, n_j$  : = label [, item-size] [, group factor]

$$\left[ \left[ \left\{ \begin{array}{c} \text{BN} \\ \text{AN} \end{array} \right\} \right] \left[ \left\{ \begin{array}{c} \text{IN} \\ \text{OUT} \end{array} \right\} \right] \left[ \left\{ \begin{array}{c} \text{FI} \\ \text{DV} \end{array} \right\} \right] \right]$$

Process I/O:



PROCOL (continued)

where:

- a means the number of channels
- b is the first channel number
- c is the repetition factor
- d is the channel number
- e is a list of identifiers
- S.U. means Symbolic Unit.

13- Are external devices and the respective transfer mechanism described explicitly in the program proper or in a system description and how does such a description look like?

ALGOL 68

Not defined in language - except that the distinction is made between binary and non-binary files.

CAMAC - IML

External devices are described explicitly by device declarations in the program proper (see 8).

Transfer mechanisms are not described explicitly, and may be different in different implementations using the same source text. It is assumed in IML that an installation may have several different implementations of the transfer mechanism, with the same logical properties but different speeds etc... These are called channels. IML recognizes that channels may be different from one another but does not define what each is. An I/O statement can specify the channel number to be used.

Example:

UBL, READ, DEVA, BUDA, REP6Ø, DA, 2

specifies a Uni-device Block transfer on LAM using the CAMAC function READ i.e. F(Ø), from the device at symbolic CAMAC address DEVA to the memory area declared as BUDA, with repetition controlled by the declaration REP6Ø and LAM declared as DA, using channel type number 2.

CORAL 66

Explicitly in the program proper.

PAS 1 - PL/I

See 12.

Example: WORD,2,DP1003,MODE(15),READYINT(5)\*

THERMO =10;

PRESS =11;

WORD,3,DP1009 NIXIE = 512.2.4;

GROUP(50) : SWITCH = 520.0.1;

Explanation:

WORD : wordwise I/O (contrary: blockwise I/O)

2 : number of device in the environment under discussion

DP1003 : Key-word for slow analog input device

DP1009 : Key-word for digital output device

MODE(n) : n is a code-number for range and time of measurement

READYINT: indicates data-ready line of slow device

THERMO : programmer-defined name for channel 10

NIXIE : programmer-defined name for four bits of channel 512, beginning at bit no.2.

GROUP : there are 50 "Switches" connected from channel no.520 on, each taking 1 bit.

PEARL

In the system-division the data-path of any terminal device is stated.

PL/I

The language only allows device description at the logical, device independent level. Direct device details must be specified external to the language.



PROCOL

External devices are described at generation time (system part) and second at the language level in the COMMON part (devices declaration and mechanism specifications). In the other compiling units, appear the proper I/O instructions.

14- Can all external device registers be addressed and accessed via I/O operations or some other control operations?

ALGOL 68

The language does not recognize the existence of external device registers.

CAMAC - IML

Yes, all external device registers can be addressed and accessed by the normal set of action statements of the language. All group-1-registers (data registers) can be input and output, all group-2-registers (status registers) can at least be tested, sensed or cleared.

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

External device registers are not directly referenced within the language.

PEARL

All external devices, which are described in the "system-division" can be addressed by the program. Special functions (like "test", "initialize") can only be performed by transferring a respective bit-pattern to the "outermost" point where those can still be regarded as normal I/O-commands.



PROCOL

External device registers are not directly referenced within the language; only 'buffers' may be read (BUF).

15- Are the external devices indexable?

ALGOL 68

This question is outside the scope of the language.

CAMAC - IML

External devices can be declared as arrays and can be accessed by subscripted variables.

CORAL 66

Not in any of our systems.

PAS 1 - PL/I

Channels may be grouped and accessed by one I/O instruction. (\*)

PEARL

Yes.

PROCOL

Channels are indexable, one or more channels may be referenced at the same time. (\*)

16- Can one single I/O instruction specify a transfer involving many external devices?

ALGOL 68

Yes, if a suitable channel is available in the standard prelude.

CAMAC - IML

IML allows multiple transfers between a set of CAMAC devices and the computer. In the case of transfers associated with data, reference of the CAMAC addresses and the internal data storage locations is on a one to one basis. There are three different multiple modes:

1. MA-mode is effectively a sequence of single mode operations using the same CAMAC function code. It is terminated on repeat count.
2. MNQ-mode terminates on q-response = 1. It is restricted to use with certain function codes used for fast testing applications.
3. MAD-mode. In this mode the CAMAC addresses are determined by the q-response from the modules (normally sequential incrementing of module addresses). The operation is terminated on reaching a specified word count or address.

CORAL 66

Not in any of our systems.

PAS 1 - PL/I

No, a group of channels must belong to one external device. (\*)

PEARL

Yes. It is possible to use a list of device-identifiers in an I/O instruction. The single identifier may also refer to a device-array.

Example: MOVE ('1'B, 7000) TO (SWITCH, ANOUT)

PROCOL

A single I/O operation may specify transfers involving several channels.

17- What is the minimum unit of data which can be transferred?

ALGOL 68

The minimum unit that can be transferred is something declared to be of mode int, real, char, bool, bits or bytes.

CAMAC - IML

Transfers are of 24-bit words within the CAMAC system and of n-bit words to and from the computer store. The number n is the specified word-length for the variable. The word-length for each variable must be a constant, not exceeding 24.

CORAL 66

Depending on the device and system, one byte. If a disk is involved some systems require the user to do the buffering.

PAS 1 - PL/I

Depends on the device. It may be a bit (process I/O), a byte (stream I/O) or a block (record I/O).

PEARL

The minimum unit of data depends on the device (its properties, including this minimum, are fixed by the implementation).

PROCOL

One bit declared as an item.

18- Up to what time is the allocation of the actual devices possible? (Programming time, compile time, link-time...)

ALGOL 68

Run-time.

CAMAC - IML

Up to run-time. Since constants can be used for the addresses, this allows allocation at programming time.

CORAL 66

Allocation is normally done at run-time.

PAS 1 - PL/I

Up to programming time.

Standard devices may be switched at run-time (see 8).

PEARL

Up to run-time.

PL/I

Various - not under direct control of language.

PROCOL

At system generation time, the connection between address of interface (or handlers) and peripheral symbolic name (or numbers), at compile time, relation between device used and its address is achieved.

19- Which possibilities are there to switch between devices or data paths at run-time?

ALGOL 68

By using mode file.

CAMAC - IML

CAMAC is essentially a multiplexing system, permitting many devices to be connected over a tree of data paths. There are extensive facilities for switching between at run-time. Each I/O instruction specifies the data path to be used, hence the device concerned.



CAMAC - IML (continued)

The data path to the I/O device can be specified as a constant, or as a single variable, or as a member of an array of variables. Multiple action instructions access numbers of devices, automatically switching between them.

CORAL 66

Depends on operating system but useful for debugging and allowed in RRE systems.

PAS 1 - PL/I

It is only possible for standard devices (see 18).

PEARL

The connection between a device-identifier and the device is made at compile time and can't be changed. But the actual device for an I/O operation is defined by the value of the device-identifier which may be a reference.

PL/I

Operating system dependent.

PROCOL

None.

20- Can virtual devices be defined (e.g. for simulation of devices by files or the use of null devices for test output)?

ALGOL 68

Proc associate enables the programmer to link a file with an array of chars held in store, rather than with an external device such as a lineprinter. All 'input/output' using that file will then be to/from that array.

CAMAC - IML

One device can be defined in place of another, but IML does not include virtual devices.

CORAL 66

This is possible on most CORAL 66 systems.

PAS 1 - PL/I

No.

PEARL

Not defined in the language report.

PL/I

Yes.

PROCOL

Not explicitly defined within the language.

21- If the allocation of actual devices is done at run-time, and the operator determines the allocation, can the language take account of this?

ALGOL 68

Outside the scope of the language.

CAMAC ~ IML

The host environment can obtain parameters from the operator which can be used in IML to control selection of device address (see 19).

CORAL 66

Depends on system again, it can be a dangerous facility but important if a test process needs upgrading to a live process.

PAS 1 - PL/I

See 8.

PEARL

Outside the scope of language.

PL/I

Operating system dependent.

PROCOL

Allocation is partly made conversationally at system generation time by the operator.

Formatting

22- What are the basic principles of the formatting mechanism used in the language (e.g. use of format strings, format procedures, format variables)?

ALGOL 68

Use of the mode format (which is associated with a formatting string). Formats can be manipulated like any other mode and may, for example, be returned from procedures.

CAMAC - IML

Not applicable: there are no code transformations or formatting in IML.

CORAL 66

Called via library procedures, hence depends on how they are written.

PAS 1 - PL/I

Format lists, containing format elements for all types of data (A, B, E, F, X, SKIP (n), PAGE, COLUMN (n), LINE (n), Picture).

#### PEARL

The transformation between the internal and external representation of data is controlled by formats. A format in PEARL is treated as an information (bit-pattern) as every other information. The formats in PEARL cover two main classes of external representation: character and graphic information.

#### PL/I

Unformatted, data directed and formatted data are all supported. The format specifications are by format strings and variables.

#### PROCOL

- Classical I/O-formats: same philosophy as in FORTRAN
- Process I/O-formats:
  - . use of subroutines associated with each function:  
PHYSCHECK, LOGCHECK, etc...
  - . use of format variable
  - . use of table of format variables.

23- How can formats be modified at run-time?

#### ALGOL 68

See 22.

#### CAMAC - IML

Not relevant.

#### CORAL 66

Altering library calls unless specifically programmed in.

#### PAS 1 - PL/I

See 22 "n" in format-items SKIP, COLUMN and LINE may be variable.

#### PEARL

A format in PEARL is comparable to a string. One can define variables of type format and may use the operations "assign" and "concatenate" with data of type format.

#### PL/I

The format string used can be changed dynamically as the field sizes, precision, etc. specified within the individual format string.

#### PROCOL

At run-time it is dynamically possible to switch from one specified subroutine of a format to another within the sequence of subroutines defined in the format (Process I/O formats only).

24- What control does the programmer have on the layout (e.g. line counting, end of line, position control, out of scale)?

#### ALGOL 68

The format facilities are similar to those provided in FORTRAN. The programmer may test the current reading/writing position by using the procedures line ended/page ended.

#### CAMAC - IML

Not relevant.

#### CORAL 66

If library procedures are not flexible enough then the programmer can write his own layouts into the program.

#### PAS 1 - PL/I

Full scale of basic format elements and software interrupts.



PEARL

The position-control, roughly spoken, enables the programmer to fix the point where character data shall be placed in case of output or where a character string (containing data) can be found.

Only those position-controls will be effected which are executable on the device which is addressed in the respective I/O statement.

PL/I

PAGE, SKIP, LINE, COLUMN and X format items are all supported as is picture format.

PROCOL

Same as in FORTRAN.

25- Which alphanumeric, graphic, etc. formats are available? ("I/O-requirements", point 1).

ALGOL 68

The character set is not defined by the language: the various components of the format string recognize items such as:

- digit
- sign
- zero
- point
- exponent
- complex
- radix
- string
- boolean

CAMAC - IML

Not relevant.

CORAL 66

Not specified in language but is currently being thought about, particularly in connection with communication systems.

PAS 1 - PL/I

For each data-type corresponding format elements. No graphic format elements.

PEARL

There are three main classes of format elements:

- mapping-control
- position-control
- string-control

The mapping-control serves to control the output of data as character string with defined properties, the input of data into given positions within the working storage. Therefore a one-by-one-correlation between the sink-elements and the mapping-controls has to be established.

Mapping-control exists for four groups of information:

- arithmetic elements
- string elements
- time elements
- free elements

PL/I

F, E, C, P, B, and A format items are available.

PROCOL

- Those used in the standard FORTRAN
- The all range of Process I/O formats.

26- Is there a possibility of automatic formatting by the system (without explicit format specification)?

If yes, how is this controlled by the type of the respective data?

ALGOL 68

Yes. Formatless output results in printing, preceeded by a space (if not at the beginning of a line), allowing sufficient positions to cope with the largest permissible value of that mode. Procedures newpage, newline, space and backspace are available for layout control. Formatless input is made using procedure read. The mode of each item required is identified and the input searched for the first character that is not a space. When it is found the required item is read in.

CAMAC - IML

Not relevant.

CORAL 66

Yes, but not controlled.

PAS 1 - PL/I

Yes, list-directed I/O. The default formats depend on the data-types.

PEARL

In PEARL the programmer has either the possibility to control his character-I/O by format elements or to rely upon free formats. This can either be indicated by completely omitting the format or by using the socalled free element. For input reasons the external representation of respective data must be equivalent to the constant-denotation of the data type of the sink. Output is always carried out in a way that all information is preserved which is contained in the data source (this means e.g. for decimal numbers that all significant digits are printed). The form of the output is determined by the precision of the data-item to be output.

PL/I

Yes, two types are supported, list and data-directed. The system makes any necessary conversions on input and on output picks the most suitable representation of the data.

PROCOL

A format has always to be defined, except for binary I/O (paper tape reader, puncher; for disk the address is needed along with the number of words to write (read)).

27- At what time is the correspondence between data types and formats established? (Compile time, link time, run-time...).

ALGOL 68

Run-time - formats may be passed around as "variables. However no operators can be defined that operate on formats and their only components that may not be determined until run-time are dynamic replicators.

CAMAC - IML

Normally compile time.

CORAL 66

Normally compile time.

PAS 1 - PL/I

At compile time.

PEARL

At run-time and dynamically.

PL/I

Depends on the compiler implementations and on options chosen by the user.

PROCOL

At run-time and dynamically.

28- Are the formats indexable?

ALGOL 68

Yes, in the sense that arrays of mode format can be declared.

CAMAC - IML

Not relevant.

CORAL 66

Not in any of our systems.

PAS 1 - PL/I

No.

PEARL

Yes, one can use arrays of formats.

PROCOL

Yes, table of format numbers may be used for a series of I/O.

29- Which possibilities of coding and decoding are there?  
I.e. which additional facilities exist to directly manipulate the internal or external code representation of data?  
(For example code conversion routines).

ALGOL 68

The procedure conv may be used to attach a different code conversion table to a file.

CAMAC - IML

Not relevant.

CORAL 66

Dependant on operating system but RRE systems allow characters to be transmitted in both coded and uncoded forms.



PAS 1 - PL/I

No language features.

PEARL

Not-formatted-not organized communication (MOVE) is the simplest form of communication in PEARL:

Binal values are moved to or from the communication register of a device. A gauge-option allows a transformation of these values before output or after input. So this statement can be used for basic control functions.

PL/I

The exact internal bit representation of any data type can be transmitted to or from a bit string. Routines are available to do table lookup directly.

PROCOL

For Process I/O, raw data and converted data are accessible by the key-words BUF and LST. All conversions are performed by standard mechanism (for Classical I/O) and by user-defined or standard conversion subroutines (for Process I/O).

Synchronization aids

3Ø- When a task calls for an I/O action, does (a) the task get suspended until the I/O has been completed or (b) the task continue?

Can the programmer specify which?

If the task continues, how can it be informed when the I/O has been completed?

ALGOL 68

- The task is suspended.
- The program cannot specify another treatment.
- Not relevant.

CAMAC - IML

I/O operations on autonomous channels (see 4) can proceed concurrently with the program. IML includes status variables to permit synchronization by the host language.

CORAL 66

Operating system feature.

PAS 1 - PL/I

- Input of process data via slow devices may be performed simultaneously.
- By enclosing the respective I/O-instructions in SIMDO; IN...; IN...END;
- Task gets suspended until each enclosed I/O-instruction is finished.

PEARL

- The task gets suspended until I/O has been completed.
- The program cannot specify another treatment.
- Not relevant.

PROCOL

- The task gets suspended.
- The program cannot specify another treatment.
- Not relevant.

31- Which features exist for explicit reservation of devices (e.g. semaphore variables)?

ALGOL 68

Sema "variables" may be used to ensure the private use of devices.

CAMAC - IML

Not applicable. There is no device reservation in IML. Any reservation must be done by the host environment.

CORAL 66

No specific features and depends on operating system but I feel it is essential for real time systems.

PAS 1 - PL/I

OPEN (filename) reserves the device connected to filename until a corresponding CLOSE (filename). Semaphore-variables may be used.

PEARL

No explicit reservation of a device on language level; semaphores and bolts may be used for that purpose.

PL/I

Devices can be reserved by the EXCLUSIVE option on the OPEN statement.

PROCOL

No explicit reservation of the device at the source level.

32- Can several tasks access the same device?

If so, how are their sequences of demands regulated?

ALGOL 68

Several tasks can access the same file, however procedure open checks to make sure that, if a file is being opened for writing then, only one task can open that file. The system action taken if this condition is violated is implementation defined.

CAMAC - IML

See 31.

CORAL 66

In most systems none as can be dangerous.

PAS 1 - PL/I

Reservation is done fully automatically with the exception indicated in 31.

PEARL

Different tasks may request access to the same device. The sequence of demands is regulated by the task's priorities.

PROCOL

Yes, where they have different symbolic name; access is handled by the handlers.

33- How is status information about I/O operations and/or devices accessible by the program? Can (or: must) it be taken into account before initiation of a new I/O operation?

ALGOL 68

The programmer can test the characteristics of a particular channel using environment enquiries: put possible, maxpage, etc... He can also test the reading/writing position of channels at any time using char number, line number and page number. It is not necessary to do this before every I/O operation.

CAMAC - IML

The IML program can access device status in several ways:

- (a) Most CAMAC actions result in a (binary) Q response being generated in the module, indicating a status (e.g. valid data or end of block). The action statement can include an escape address to jump to depending on the value of Q. The Q may be tested by a subsequent statement.
- (b) Most CAMAC modules have internal status registers which can be sensed by particular CAMAC actions (TLAM = testlook-at-me; TSTAT = test status).
- (c) devices driven by CAMAC module usually give status information to the same module (at a different subaddress or group from the normal data). This can be accessed by an explicit CAMAC action.



CAMAC - IML (continued)

- (d) Modules can autonomously generate an L (or Look-At-Me) signal. Depending on crate-branch-system organization, this can interrupt the program or set a status bit in the higher level, which can be sensed by an explicit branch or system action.

The status may (but need not) be taken into account before initiating a new I/O operation by use of the appropriate statement. The status of activity in an IML program is available in status variables which can be used by the host program.

CORAL 66

Status information can be returned as the answer to a library procedure.

PAS 1 - PL/I

No language features.

PEARL

Standard responses which may occur during the execution of an I/O statement are not yet specified.

PL/I

Event variables can be used to signal status of I/O operations but the language places no restriction on the number of simultaneous outstanding requests to the same device. (NOTE: Generally there is a pragmatic constraint which is operating system dependent).

PROCOL

Not possible at the language level.

Safety measures concerning I/O

- 34- Protection against programming errors: How far can I/O statement be checked for logical correctness before run-time? (For example type matching between internal and external representation of data).



ALGOL 68

Compile-time checking of types in I/O procedure calls is carried out.

CAMAC - IML

The structure of IML permits normal translator checks, viz:

- Correct number of arguments
- Correct arrangement of external and internal references
- Legal hardware address (where explicit)
- Correct association of internal reference, external reference and statement type.

Specific implementations of IML will not necessarily guarantee these checks.

CORAL 66

Fully checked by compiler.

PAS 1 - PL/I

Compiler checks correspondence between data- and format-type.

PEARL

A compile-time check for type-matching is only possible for data and formats.

PL/I

Most STREAM I/O statements will execute successfully if they are syntactically valid. It is somewhat more likely that RECORD I/O statements can be invalid despite having valid syntax, however, most such errors are due to discrepancies between the required file description and the actual file description and are caught when the file is OPENed.

PROCOL

Compile-time check for type of data is performed.

35- Which file protection measures exist?

ALGOL 68

An identification string must be supplied to procedure open before any file can be opened. There are no other concepts of file protection existing in the I/O language.

CAMAC - IML

IML does not include the concept of files. This is considered to be the province of the host environment.

CORAL 66

Operating system feature but should be secure.

PAS 1 - PL/I

No file protection except by use of semaphores.

PEARL

A data set can be protected against writing by the file-integrity statement. The write protection which is enacted by LOCK is removed by UNLOCK.

PL/I

Individual records can be locked while being accessed by a task. EXTERNAL variables can be used to control simultaneous use of files. Data can be readily encrypted.

PROCOL

Not in the language (MMT).

36- Which memory protection ( working storage) method does the language reflect (e.g. separation of code and data)?

ALGOL 68

The language implies the separation of code and data into separate areas due to the dynamic nature of ALGOL 68 storage.

CAMAC - IML

Not applicable (depends on implementation). IML allows segmentation of data and code so that protection is possible if the host environment has protected segments.

CORAL 66

Complete separation of code and data with the possibility of read only protection on code and constants.

PAS 1 - PL/I

Software interrupt for SUBSCRIPT RANGE,  
semaphore variables.

PEARL

Implicitly the same methods as usual with compiler languages.

PL/I

The language is functionally a von Neumann language but otherwise implies no other memory protection model.

PROCOL

Not in the language.

37- Are there other (special) protection mechanisms (e.g. against exceeding of buffer limits)?

ALGOL 68

Buffer limits may not be exceeded by the programmer - the I/O procedures ensure this.

CAMAC - IML

- External devices can signal when a hardware limit is exceeded.
- Arrays of external addresses may have bound checks.
- An internal reference can be to a buffer which can signal when it is exhausted.
- In reference to buffers: the element accessed is always implicit, so there is no possibility of referencing outside the buffer.

CORAL 66

No.

PAS 1 - PL/I

The programmer may define whether overflow-, zerodivide- and subscript-range-checks shall be performed at run-time.

PEARL

No other explicit protection methods.

PL/I

Most implementations of the language do use special protection mechanisms. One implementation (the Checkout compiler) explicitly traps all known violations of the language.

PROCOL

Not explicitly in the language.

38- How can the addressing of non-existent devices be prevented?

ALGOL 68

Yes- by procedure open - although as defined it does not indicate whether the opening was successful or not.

CAMAC - IML

Addressing of non-existent devices cannot be prevented (since it is run-time dependent), but it can be detected. The CAMAC hardware includes an X-response for this purpose, which is available to the operating system. IML includes statements to enable and disable this facility.

CORAL 66

Outside the scope of the language.  
Operating system checks.

PAS 1 - PL/I

Addressing cannot be prevented. Operating system will print out an error message.

PEARL

Outside the scope of the language.  
Operating system checks.

PL/I

The requisite OPEN fails.

PROCOL

This is not possible, since devices are given a symbolic name.  
Operating system checks.

39- Which activities are initiated in case of a device failure?

ALGOL 68

Associated with each file are procedures: physical file end, format end, value error, char error and other error (not defined in the report). The user may assign to these procs to define his own error recovery.



CAMAC - IML

There can be an implementation-dependent action in the case of X-error; the program can test for device failure (using X-response, Q-reponse, or a status bit) and instigate some appropriate action.

CORAL 66

Depends on operating system which would normally give some error value to the program.

PAS 1 - PL/I

Reaction of operating system.

PEARL

Not defined in the language. Depends on operating system.

PL/I

Many device errors will generate a signal to an optional piece of user code.

PROCOL

If the I/O has been preceded by a WAIT a warning is issued by the system.

40- Possibilities of intervention by the operator: which kinds of messages are given to the operator in critical situations and which possibilities of intervention does he have?

ALGOL 68

Not defined in the language.

CAMAC - IML

Nothing automatic. All interventions have to be explicitly programmed.

CORAL 66

Not defined in language.

The operating system will tell the operator about device faults, etc... for normal devices with the applications program handling more unusual peripherals.

PAS 1 - PL/I

Message will be given to the operator. His commands may be interpreted by programs.

PEARL

Not defined in the language.

PL/I

The language has a DISPLAY with REPLAY capability for communication with the operator. Other capabilities also exist but are operating system not language features.

PROCOL

There are possibilities of intervention by the operator when he receives any types of system warnings (MMT).

I/O-error-handling

41- Is there an error handling mechanism defined in the language and how does it work?

ALGOL 68

Only for I/O operation. See 39.

CAMAC - IML

CAMAC hardware can detect certain errors (see 38, 39). Statements in IML allow the programmer to specify a label to go to whenever such an error is detected.

CORAL 66

No.

PAS 1 - PL/I

Yes, in case of conversion-, zerodivide-, over/underflow-, transmit- and subscriptrange-errors programmer defined statements may be performed.

PEARL

Standard responses which may occur during the execution of an I/O statement are not yet specified. In case they are specified the response-statement enables the programmer to overwrite system responses and to specify actions according to his own requirements.

PL/I

Yes, there is a generalized error handling mechanism which allows the user to code ON-blocks. He can define which errors the blocks are to be activated upon and can dynamically change the range for which these blocks are active. He can also dynamically change the ON-block for a given error.

PROCOL

Not defined in the language. But exist within the system (MMT).

42- What possibilities exist for programmed error reaction (e.g. addressing a non-existent device)?

ALGOL 68

Several 'on' procedures are provided for the programmer to associate an error-recovery procedure with a particular error condition. For example:

on logical file end,  
on physical file end, etc...

In addition procedure open returns an integer which, if non-zero indicates the reason why the file cannot be opened.

CAMAC - IML

Program can test for error (e.g. X-error meaning non-existent device or Q response or status) and take appropriate action. In some CAMAC - IML instructions the addressing of a non-existent device is not an error, but a termination condition (e.g. address scan: a multiple device action can operate on consecutive hardware addresses, terminated without error when an address is reached from which no response is received).

CORAL 66

On some systems it is possible to trap errors from the O.S. and provide full use recovery as necessary.

PAS 1 - PL/I

See 41.

PEARL

Asynchronously to the activities of the system interrupts from the process or any device may arrive. An interrupt leads to the activation of an additional task if there is any scheduled for this interrupt.

PL/I

As for 41.

PROCOL

Not explicitly defined within the language.

43- What device status information ( e.g. raised by the O.S.) can be specified and/or tested by the program?

ALGOL 68

The only possibilities are the procedures mentioned in 39.

CAMAC - IML

See 33.

Status information which can be tested by the program includes:

- device present
- function recognized
- valid data present
- end of block of data
- LAM source (s) enabled

The program can specify which LAM sources are to be enabled.

CORAL 66

Implementation dependent but is usually done via a special op.-  
syst.call or as parameters returned from an I/O transfer.

PAS 1 - PL/I

None.

PEARL

Every message may be treated as interrupt or as signal.

PL/I

Almost all I/O and computational errors.

PROCOL

None.

44- On what kinds of I/O errors can programs be interrupted?

ALGOL 68

Not defined in the language.

CAMAC - IML

The normal sequence of execution can be interrupted and the current I/O transfer terminated as a result of:



CAMC - IML (Continued)

- completion of specified number of words in transfer
- buffer full/empty i.e. internal limit
- end of block, i.e. external limit
- errors status (X-response) from device.

The program must then decide whether this situation is to be considered to be an error, and select appropriate action.

CORAL 66

Implementation dependent but the current trend is not to allow an applicator program to be interrupted with messages, etc. queued up by the op syst which also does time outs, etc. on an application program that has gone wild.

PAS 1 - PL/I

See 41.

PEARL

Not specified in the language.

PL/I

See 43.

PROCOL

Out of the scope of the language.

45- How can errors be used as starting conditions for tasks?

ALGOL 68

In I/O operations by making procedures physical file end, value error, etc. initiate other activities.

CAMAC - IML

An error in a module can be made to generate a LAM in the module. A LAM can raise a program interrupt (see 44). IML allows the program to associate a label with an interrupt source. Thus an error can cause the program to start at a given label. IML does not have explicit task structure. (This is the province of the host environment).

CORAL 66

Operating system feature.

PAS 1 - PL/I

See 41.

PEARL

If they can raise an interrupt, by the normal interrupt handling mechanisms.

PL/I

By executing the TASK activation statement in the requisite ON-block.

PROCOL

Not directly possible within the language.

46- After which errors is it principally possible to continue the program which was affected?

ALGOL 68

After any error condition for which an error procedure is defined. See 39.

CAMAC - IML

All. (Error exits must be provided by the programmer).

CORAL 66

Varies too much to comment on.

PAS 1 - PL/I

See 41.

PEARL

Not specified in the language.

PL/I

If the ON-block has removed the source of the error, the program can continue in execution.

PROCOL

When the error does not provoke the abortion of the task.

Graphic I/O

47- In which way is graphic I/O handled within or above the language?

ALGOL 68

There are no graphic I/O facilities in ALGOL 68.

CAMAC - IML

No graphic in related elements in IML, but of course elements and modules can be used with graphic devices.

CORAL 66

Not in the language but handled via packages.

PAS 1 - PL/I

47-54 - Not relevant, as there is no graphic I/O.

PEARL

There are special statements for graphic I/O in the language:

DRAW source TO sink FORMAT graph-format;

SEE sink FROM source FORMAT graph-format;

PL/I

There are no specific provisions for "graphic I/O" devices in the language but the language can be readily used to format complete displays and output them to the required device.

PROCOL

There are no graphic I/O facilities in PROCOL.

48- Which data types can be output as graphic elements?

PEARL

One has to distinguish between true graphic I/O and just a form of character I/O via graphic devices. The latter is performed by a character-communication-statement for a device capable to display character strings. By the first only data declared with an integer - or real - attribute can be handled.

49- How can graphic and character I/O be combined on graphic devices?

PEARL

All character-nomapping-control elements which are applied to the respective device by a character-communication-statement effect the display of characters in the same way as they would do in case of true character I/O.

50- Out of which basic elements are graphic structures constructed?  
(For example points, vectors).

PEARL

There is a free choice for the programmer to use points, vectors or polynomial interpolation.

51- Which possibilities exist on language level for the representation of more elaborate structures (e.g. curves)?

PEARL

One can indicate that a new picture shall be started and all information contained in the following I/O statements within one task is drawn into the same picture until explicitly a new picture is started.

52- Are there fixed, prefabricated graphic elements (e.g. polygon, coordinate system) which can be output by simple control instructions?

PEARL

Not defined in the language.

53- How are layout-functions controlled on language level (e.g. brightness, scale, offset)?

PEARL

Special control elements allow to demand the interpretation of data items as brightness or colour. The interpretation depends on the special implementation.



54- Are there graphic input features? If so, how do they work?

PEARL

The graphic input statement is:

SEE sink FROM source FORMAT graphic-format.

The language defining report contains nothing about how it works.

Binary I/O

55- From (to) which external devices can binary data be received (sent)?

ALGOL 68

Binary data can be transput to any channel for which the environment enquiry bin possible is true (see report 10.5.1.1 h).

CAMAC - IML

All CAMAC devices.

CORAL 66

The operating systems in use at RRE allow binary data to be sent to any relevant device, i.e. not printers in general.

PAS 1 - PL/I

From/to all external devices.

PEARL

Generally from/to all external devices (implementation dependent).

PL/I

All STREAM I/O devices can transmit BIT strings.

PROCOL

From/to adequate peripherals.

56- Is it possible to read or write indifferently on the same file formatted and binary data (e.g. for buffering of alphanumeric or graphic data)?

ALGOL 68

Not possible.

CAMAC - IML

Not relevant - IML has neither formatting nor file handling.

CORAL 66

The operating systems allow buffering of information.

PAS 1 - PL/I

Not possible.

PEARL

Yes, possible.

PL/I

Yes, possible.

PROCOL

Not possible.

Process I/O ("I/O-requirements", point 3)

57- What are the differences between process I/O and conventional I/O in the language in question?

ALGOL 68

There are no process I/O facilities in ALGOL 68.

CAMAC - IML

No conventional I/O present.

CORAL 66

None.

PAS 1 - PL/I

Different key-words and different structure of commands.

PEARL

Not-formatted-not-organized communication is the simplest form of communication in PEARL. The MOVE statement can transport data in binary form without any transformation.

PL/I

APG/7 implemented some special language which is not properly part of PL/I, the SCAN and SET statements and the ANALOG, DIGITAL, COUNTER and PULSE attributes. These features are for process I/O only. All answers in this section are based on APG/7.

PROCOL

The differences between conventional I/O and process I/O are only concerned with the category of conversions which are to be performed along with the data transfer and specified in the format.

Apart this, they are very similar to the conventional ones in their use and formulation.

- Definition of the peripheral is made in the common block and is valuable for the whole application.
- Format specification is made in the task body and is valuable for the whole task.
- I/O calls are done in the same way as they are in FORTRAN.

58- Which classes of process I/O are there in the languages?

ALGOL 68

Not relevant.

CAMAC - IML

Single actions, uni-device block transfers, and multi-device actions (see 42).

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

See 1, 2.

PEARL

The above mentioned MOVE is the only way to communicate with process-equipment.

PL/I

ANALOG, DIGITAL, COUNTER and PULSE.

PROCOL

The class of process I/O is unique (see following question).

59- Which classes of external devices are considered (and handled) as process peripherals?

ALGOL 68

Not relevant.

CAMAC - IML

All CAMAC devices.

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

Digital-I/O, analog-I/O (slow, medium, fast), interrupt input, alarm unit (dynamic digital input), counter I/O.

PEARL

All possible peripherals may be considered as process peripherals if the user wishes to do so.

PL/I

Digital and analog input/output devices.

PROCOL

Three classes of peripherals are considered:

- analog
- digital
- special

60- Which data types (internal representation) can be input or output via process I/O?

ALGOL 68

Not relevant.

CAMAC - IML

Fixed length bit-strings (words).

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

Subscripted and non-subscripted "Channel"-variables of size 1-16 bits.



PEARL

Binary data. Other data must be converted before output or after input to/from binary representation (by GAUGE procedure).

PL/I

Arithmetic and bit strings.

PROCOL

- Digital I/O : type is any identifier and subscripted variable (REAL - Type excluded)
- Special I/O : none; only a standard address is specified
- Analog I/O : simple I/O: as for digital I/O  
multiple I/O: subscripted variable (not REAL).

61- Which data aggregates can be input or output via process I/O?

ALGOL 68

Not relevant.

CAMAC - IML

The only data aggregates in computer memory recognized by IML are one dimensional arrays and buffers; the only type is binary. Data aggregates in external devices can be either sequential at one device (e.g. plotter, disk) or spread over a set of devices.

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

Arrays of data.

PEARL

All data aggregates may be used as sink or source.

PL/I

Arrays.

PROCOL

Tables of variables (non REAL type) can be output or input.

62- Which kinds of process data ( external representation) can be input or output? (For example analog signals, digital signals, pulses, bit-patterns, etc.).

ALGOL 68

Not relevant.

CAMAC - IML

By using appropriate CAMAC modules, data words can represent digital pulses, bit-patterns, integers, characters, real numbers, logical values, or digitized analog signals.

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

See 59.

PEARL

That is not a question to be answered in the language. It depends only on the peripheral devices of a particular installation.

PL/I

As for 58.

PROCOL

- Analog signals
- Digital signals
- Bit-patterns.

63- Which are the formatting facilities concerning process data?

ALGOL 68

Not relevant.

CAMAC - IML

The external wordlength in the CAMAC system is 24 bits. The internal (computer) wordlength for a variable can be specified in declaration statements. For wordlengths which are less than 24 bits, there is automatic packing/unpacking. There are no language elements for general packing/unpacking or other formatting of data.

CORAL 66

Depends on system implemented.

PAS 1 - PL/I

See 6.

PEARL

Modification of transferred data is only possible by "GAUGE" option.

PL/I

None specifically required.

PROCOL

All process data can be transferred via a sequence of conversions specified within the format (as for conversational I/O).

The conversion mechanisms available include:

- (in input only) the physical control of the data (PHYSCHECK)
- The filtering of the data (FILTER)
- Conversion (CONV)
- Logical control of the data (LOGCHECK)

For each mechanism the user specifies one or several special sub-routines (written in PROCOL by the user) which are called when the format is processed.

64- Which are the calibration mechanisms? ("I/O-requirements", point 3.4).

ALGOL 68

Not relevant.

CAMAC - IML

Nothing in IML. Calibration can be programmed explicitly.

CORAL 66

Depends on operating system.

PAS 1 - PL/I

See 6.

PEARL

If data have to be converted, coded, decoded, or calibrated, the gauge-procedure has to be applied on these data.

PL/I

User coded.

PROCOL

Calibration mechanisms are included in the format. (FILTER, CONV, LOGCHECK).

65- How can process I/O statements be applied to non-process peripherals?

ALGOL 68

Not relevant.

CAMAC - IML

Since there is no distinction between process and non-process, the question does not arise.

CORAL 66

Via stream switching in the operating system.

PAS 1 - PL/I

Not possible.

PEARL

In the same way process peripherals are handled.

PL/I

Not so.

PROCOL

By use of the special type.

66- How can process-peripherals be simulated via non-process peripherals?

ALGOL 68

Not relevant.

CAMAC - IML

It is not possible to do this using only language elements of the CAMAC - IML.

CORAL 66

Using a simulator program and stream switching.

PAS 1 - PL/I

Not possible.

PEARL

The MOVE-statement can also be applied to "non-process" peripherals.



PL/I

Not so.

PROCOL

Not useful.

67- How are process interrupts handled?

ALGOL 68

Not relevant.

CAMAC - IML

Each device can call for attention by causing its driver module to assert its LAM. The crate controller can either deal with it directly or make a demand to the branch controller. Typically the branch controller reacts to a demand by sensing crate controllers in parallel; it gets a "Graded L Pattern" of 24 bits indicating which LAMs are present, then interrupts the processor.

IML allows a label to be associated with each particular graded L. When the interrupt happens, control is transferred to the appropriate label.

Masks are present to enable and disable the path for LAM signals at various stages, from module to branch driver.

CORAL 66

Depends on operating system.

PAS 1 - PL/I

Events are connected with process interrupts in the "EVENTS"-part of PAS 1-programs. An interrupt causes the event to be "set" and the statements which are connected with the event in the "ACTIONS"-part of the program are performed as a task.

PEARL

As all other possible interrupts, they may be connected with an ON-statement to a task.

PL/I

By system build time allocation of tasks to process interrupts.

PROCOL

Not specified in the language.

68- How are arrays of process interrupts handled?

ALGOL 68

Not relevant.

CAMAC - IML

The "Graded-L" mechanism described in 67 deals with arrays of interrupts. If several interrupt sources are mapped onto the same graded L bit, then the IML program must investigate. There exists a special access mode in IML and CAMAC, the MNQ mode, for fast detection on interrupt source.

CORAL 66

Depends on operating system.

PAS 1 - PL/I

As arrays of events.

PEARL

Elements of interrupt arrays are treated like single interrupts.

PL/I

See 67.

PROCOL

Not specified in the language.

69- How are sources of process-interrupts identified?

ALGOL 68

Not relevant.

CAMAC - IML

A single CAMAC access to a module can sense whether or not it is asserting LAM. The MNQ mode in IML allows a single statement to search an array of devices, and find the first which is asserting LAM.

CORAL 66

Depends on operating system but normally as replies to system calls.

PAS 1 - PL/I

By connected events (see L 67).

PEARL

The connection between an interrupt and a device may be described in the "system-division" of the PEARL-program.

PL/I

See 67.

PROCOL

Not specified in the language.

70- How is I/O of high data rates managed?

ALGOL 68

Not relevant.

CAMAC - IML

A single statement in IML can call for the transfer of a sequence of words between the computer memory and either one CAMAC device or an array of devices. High data rates can be achieved by synchronizing the successive transfers by LAM signals from the module, or even by transferring the block of words without individual synchronizing signals from the module.

The action may be carried out by an autonomous channel, e.g. direct memory access or external memory device.

CORAL 66

Normally by using an array and handing the address of the first item and the length to the operating system.

PAS 1 - PL/I

By blockwise I/O (possible DMA).

PEARL

Not language dependent.

PL/I

By the normal process input/output statements.

PROCOL

By use of table of formats.

71- How can block transfer of process data be handled?

ALGOL 68

Not relevant.

CAMAC - IML

A single statement in IML can transfer a block of data in the following modes: UBL, UBS, UBC.



CAMAC - IML (continued)

UBL-mode and UBS-mode are uni-device block transfers, in which the module sets a LAM when it is ready to transfer a word.

In UBL-mode the LAM is connected normally and two CAMAC cycles are needed for each word transfer: one to read the LAM and the other to transfer the word and clear the LAM. In UBS-mode the LAM is connected specially to avoid the need to read the LAM. Thus only one CAMAC cycle is needed for each word transfer.

UBC-mode is a uni-device block transfer in which transfers are synchronized by the system controller.

CORAL 66

As for 7Ø.

PAS 1 - PL/I

See 7Ø.

PEARL

Not specified on language level.

PL/I

See 7Ø.

PROCOL

Either directly by use of the same format for each element of the table or by use of a table of formats, each format for each element.

File-handling ("I/O requirements", point 6)

72- How is the declaration of files and the installation of data sets managed, and at what time is it done?



ALGOL 68

Files are declared by declaration of the form

file a, b, c;

Files are, in fact, structures some fields of which may be examined by the users and others are 'secret'. The association of a file with data on a hardware devices is done in the open procedure.

CAMAC - IML

There is no file handling in IML. (This is the province of the host environment).

CORAL 66

Operating system dependent.

PAS 1 - PL/I

Declaration of files at compile-time.

PEARL

The usual declare-sentence as used to declare any identifier for any data also serves to declare identifiers with attribute file. A data set of a defined size on a given device is created and named by a CREATE-statement.

PL/I

Files can be declared in the language but data set creation time is operating system and implementation dependent.

PROCOL

No file handling in PROCOL.

73- Which operations on files can be performed (e.g. open, close, delete, protect)?

ALGOL 68

open	close
create	lock
establish	scratch
set	
reset	

See the Report 10.1.5.2.

CAMAC - IML

Not relevant.

CORAL 66

Operating system dependent but normally deletion can only be done via an applications program.

PAS 1 - PL/I

Explicit file operations: OPEN/CLOSE.

PEARL

Files may be opened, closed and deleted. They may be locked or unlocked.

PL/I

OPEN, CLOSE, READ, WRITE, GET, PUT.

PROCOL

Not relevant.

74- Which kinds of access to files exist (e.g. random, sequential)?

ALGOL 68

Random and sequential.

CAMAC - IML

Not relevant.

CORAL 66

Normally only sequential.

PAS 1 - PL/I

Sequential.

PEARL

This is device-dependent, but if possible, a random access is possible as well as a sequential access.

PL/I

Sequential, Direct and Transient.

PROCOL

Not relevant.

75- What is the addressing structure of a file (e.g. block structure, linear structure)?

ALGOL 68

Files are treated as being block structured with, in ALGOL 68 terminology,  $c$  characters per line,  $l$  line per page and  $p$  pages in the file. Of course, the file can be made to have a linear structure (e.g. a paper tape) by making  $c$  very large and  $l=p=1$ .

CAMAC - IML

Not relevant.

CORAL 66

Normally linear structure.

PAS 1 - PL/I

There are linear structured files (PUT/GET) and block structured files (READ/WRITE).

PEARL

A source-file or a sink-file is described by a file name and so called file coordinates. The file coordinates are: page-number, line-number and character number.

PL/I

Stream and record.

PROCOL

Not relevant.

76- What is the smallest addressable unit of a file? Is it smaller than a hardware dependent unit (e.g. block)?

ALGOL 68

The smallest addressable unit is the character.

CAMAC - IML

Not relevant.

CORAL 66

Normally one byte or character.

PAS 1 - PL/I

One character in case linear structured files, one block of choosable size in case of block structured files.

PEARL

The smallest addressable unit is the so called "char". This is an implementation dependent representation of character-, graphic- or binary information. The relationship to the hardware structure is not defined.

PL/I

1 bit.

PROCOL

Not relevant.

77- Which classes of data can data sets consist of (e.g. binary, alphanumeric)?

ALGOL 68

Both binary and character.

CAMAC - IML

Not relevant.

CORAL 66

Any.

PAS 1 - PL/I

All types of data.

PEARL

They may consist of all possible data.

PL/I

All data types.

PROCOL

Not relevant.

78- Are files addressed by numbers or by symbolic names?

ALGOL 68

Files are addressed by symbolic name.

CAMAC - IML

Not relevant.



CORAL 66

By symbolic names in RRE systems.

PAS 1 - PL/I

Symbolic programmer defined names.

PEARL

Files are addressed by identifiers.

PL/I

Names.

PROCOL

Not relevant.

79- Is the status of a file accessible by the program (e.g. file protection, length, class of data)?

ALGOL 68

Files are all connected to a particular channel. The characteristics of that channel can be investigated using the environment enquiries, for example: bin possible, maxline, put possible, etc.

CAMAC - IML

Not relevant.

CORAL 66

No, but it should be.

PAS 1 - PL/I

Partly by software interrupts, ON ENDFILE, ON TRANSMIT.

PEARL

May be accessed by implementation dependent functions.

PL/I

Not required.

PROCOL

Not relevant.

80- Can files be declared as virtual devices and hence be handled by the respective I/O instructions?

ALGOL 68

This may be possible in the revised report.

CAMAC - IML

Not relevant.

CORAL 66

Yes, in the latest RRE system.

PAS 1 - PL/I

No.

PEARL

Yes, possible, since files may consist of any kind of data.

PL/I

All files are device independent.

PROCOL

Not relevant.

#### Special I/O-facilities

81- Is there any facility for telecommunication and how does it work?

ALGOL 68

No.

CAMAC - IML

Nothing in IML, but of course a module can drive a telecommunication line and it can be accessed in the usual way.

CORAL 66

No, but work is planned in this area.

PAS 1 - PL/I

No.

PEARL

The MOVE statement may be used for telecommunication if the implementation provides supporting facilities.

PL/I

Yes, uses RECORD I/O statements on TRANSIENT files.

PROCOL

No.

82- How is date and time handled by the language?

ALGOL 68

There are no facilities mentioned in the report although procedures could be provided in the standard prelude.

CAMAC - IML

No special features - if required use module (e.g. Borer 141) to generate them and read module in the usual way.

CORAL 66

Via operating system calls.

PAS 1 - PL/I

Time and date accessible by language instructions.

PEARL

The hours of CLOCK data are interpreted modulo 24.

External representation of DURATION-type data :

e.g. 53HRS28MIN0.00SEC

of CLOCK-type data:

e.g. 8:30:30:01

PL/I

By specific functions.

PROCOL

Date is not handled by the language.

Time is handled by means of timers to which a specific time period has been defined.

83- Which data, events, schedules etc. can be accessed by the operator and how is this done?

ALGOL 68

The question is outside the scope of the language report.

CAMAC - IML

Not applicable to IML.

CORAL 66

Normally none unless the applications program is specifically written to do this.

PAS 1 - PL/I

All variables can be declared to be accessible via name or compiler-assigned number and to be changeable or not.

PEARL

Out of scope of language definition.

PL/I

All, by user procedure.

PROCOL

(\*)

84- How can data be exchanged between external devices (e.g. directly or via memory buffer)?

ALGOL 68

Only by use of read/write procedures via memory buffer.

CAMAC - IML

Not directly in IML - only via memory buffer by block, single or multiple transfers.

CORAL 66

Depends on operating system and hardware.

PAS 1 - PL/I

Via memory buffer.

PEARL

The MOVE-statement can be used for direct device-device transfer.

PL/I

This is an extra language question.

PROCOL

By use of 2 I/O orders.



85- How can unformatted data be transferred between working storage and external memory or between different CPUs in a computer network? ("I/O-requirements", point 2).

ALGOL 68

The question is outside the scope of the language report.

CAMAC - IML

Assuming that the connection in hardware is by CAMAC, then simply by action on the appropriate module. Otherwise IML does not apply.

CORAL 66

Depends on operating system and hardware.

PAS 1 - PL/I

By use of files.

PEARL

Not explicitly defined in the language report.

PL/I

This is an extra language question.

PROCOL

Out of the scope of the language.

86- Are there any other special I/O facilities that are not mentioned above?

ALGOL 68

No.

CAMAC - IML

Yes:

- Synchronization of words input/output for multiword transfer.
- Specification of demand sources (interrupt stimuli).
- Specification of sequences of hardware addresses for use in multiple transfers.
- Controlling crates and branches (initiate, clear, test/enable/disable crate demand, enable/disable branch demand, enable/disable Xerror trap).

CORAL 66

No.

PAS 1 - PL/I

The operator can start so called CONSOLE-procedures, whereby he may provide this procedure with parameters.

PEARL

No.

PL/I

No.

PROCOL

No.

-151-

ALGORITHMIC

PRECEDING PAGE BLANK-NOT FILMED

General

- 1- Give a brief history and bibliography.

ALGOL 68

Working group 2.1 on ALGOL of IFIP has discussed the development of "ALGOL X", a successor to ALGOL 60, since 1963. A succession of meetings was held between 1963 and December 1968 each producing a number of approximations to the language which became published as ALGOL 68. Since publication a large number of revisions to the language have been considered and the revised report is expected to be published late in 1973.

References:

1. A. van Wijngaarden et al. "Report on the Algorithmic language ALGOL 68" Numerische Mathematik 14 (1969) 79-218.
2. P.M. Woodward and S.G. Bond. "ALGOL 68-R User's Guide" HMSO.

CORAL 66

CORAL 66 was designed at the Royal Radar Establishment, Malvern, England in 1966 as a language which was suitable for use in process control and real time systems. It is partly derived from JOVIAL and ALGOL 60 and is intended to produce efficient object code. About 1970 it was adopted by the Ministry of Defence as a standard for weapon systems and "The Coral Group" was formed in 1973 to promote its use on a national basis within the United Kingdom, under the technical authority of the Royal Radar Establishment.

Reference:

B. Gorman, P.R. Wetherall, P.M. Woodward  
Official Definition of CORAL 66  
Her Majesty's Stationary Office



#### PAS 1 - PL/I

PAS1-BBC\_PL/I-Subset is a compound system of two languages. PAS1 serves mainly the problems of the process, i.e. I/O of process data, system description and scheduling, PL/I-Subset serves the arithmetic and the standard I/O-problems.

The development of this language system was started 1968, because at that time there was no high level language for process control computers available. PL/I was chosen as a base, because it provided more features necessary for realtime applications than other languages.

#### PEARL

The language "PEARL" (process and experiment automation real time language) has been developed by a group of approx. 13 representatives of vendor companies, users, software houses and institutes in the FRG. This group was founded in 1969 and published a first concept (1) in 1970.

A detailed language description was published in April 1973 (2) and an introductory paper in September 1973 (3).

The work of the PEARL-group was sponsored by the German Ministry of Research and Development either directly or via the "Project PDV" (= Prozessdatenverarbeitung).

PEARL is a "middle level language" (like e.g. FORTRAN, ALGOL or PL/I and intended to be used by e.g. the process engineer, physicist, chemist with little experience in programming.

#### References:

1. PEARL, The Concept of a Process and Experiment-oriented Programming Language; elektronische Datenverarbeitung, 17 (1970), 429-442.
2. PEARL, A proposal for a process- and experiment automation real time language; KFK-PDV1, April 1973.



PEARL (continued)

3. PEARL, eine Prozess- und Experiment-orientierte Programmiersprache; Angewandte Informatik 9/73, 363-372.

PL/I

(\*)

PROCOL

In 1970 the French government Automation Committee in D.G.R.S.T. sponsored the PROCOL project. This action was taken to fulfill the need of the Process Control community users. STERIA is responsible for the implementation of the language on different machines and for different users. Several applications have been successfully designed and programmed in PROCOL and this can stand as a living proof of the fitness of the language to the needs of the users.

Bibliography:

"PROCOL: Process Control Language"

M. RITOUT, P. BONNARD, P. HUGOT

5<sup>th</sup> IFAC Congress, Paris, June 1972.

"The PROCOL Language"

G. LOUIT, M. OSSOLA

DATAFAIR, Nottingham, April 1973.

"PREX, the hierarchical, modular, flexible Real Time executive  
"of PROCOL"

P. HUGOT, M. OSSOLA

3<sup>rd</sup> European Seminar on Real Time, Ispra, May 1973.

"Development and Implementation of PROCOL"

P. HUGOT

INFO Symposium on Real Time Computing, London, October 1973.

"PROCOL on MITRA 15" : Extensions and Efficiency

J.B. FRANK

4<sup>th</sup> European Seminar on Real Time, Budapest, March 1974.

PROCOL (continued)

"PROCOL a high level programming system for Real Time applications"

M. RITOUT, P. HUGOT

6<sup>th</sup> Australian Computer Conference, Sydney, May 1974.

RTL/2

RTL/2 was developed at the Corporate Laboratory of Imperial Chemical Industries Ltd.

The work had its origins in the successful development of a simple variant of ALGOL on a Ferranti Argus 400 in 1967 for the control of laboratory processes. As a consequence a formal project was started in 1969 to develop a more general language and related basic software for process control and similar on-line applications. The work was undertaken in stages with RTL/1 being a prototype. experience with RTL/1 led to the development of RTL/2. A formal specification of RTL/2 was first published in 1971 and a revised final version in 1974.

References:

Barnes, J.G.P., Real-time language for Process Control.

Computer Journal, vol 15, pp 15-17.

An Introduction to RTL/2. Corporate Laboratory, ICI.

RTL/2 Language Specification. Corporate Laboratory, ICI.

2- For which class of processing problems was the language explicitly designed?

Find quotations from original reports.

ALGOL 68

'ALGOL 68 is designed to communicate algorithms, to execute them efficiently on a variety of different computers, and to aid in teaching them to students'. (Report 0.1 (b)).

There is no explicit statement in the Report on the class of processing problems to which ALGOL 68 is addressed but it may be

#### ALGOL 68 (continued)

presumed that it is intended for both numerical and non-numerical applications. Facilities exist within the language for examining the characteristics of the computer (word length, mode bits, etc.) used and it may therefore be assumed that the language is intended for system programming applications.

#### CORAL 66

CORAL 66 was designed for use in process control and real time systems. It is particularly suited to areas of application where object code efficiency is of importance.

#### PAS 1 - PL/I

All applications of process control computers should be served. PAS 1 - PL/I was not developed for special purposes.

#### PEARL

Chapter 0.3 of the PEARL-Report starts with the sentence:

"Being a general-purpose realtime language, PEARL contains a quite complex set of task operations and mechanisms for timing and scheduling."

Further on one can read: "It is possible to describe completely even complex hardware configurations, to use symbolic names for hardware entities throughout the whole problem programs and to connect hardware interrupts to their software responses."

"PEARL contains possibilities of introducing user-defined data types and operators with restrictions for reason of efficiency.

#### PL/I

(\*)

#### PROCOL

The language has specially been designed for Real Time and Process Control Applications. The following running applications have been programmed in PROCOL :

PROCOL (continued)

- Control of the French telephone traffic in a switching centre.
- Simulation of an arm-system for military applications.
- Data collector and control for a gas-factory.
- Control of a distillery column process.
- Control of a polyethylene fabrication process.

RTL/2

RTL/2 is designed for use in industrial or research computing and is particularly suited to computer applications involving real time data acquisition and control. These may typically include industrial process control, traffic and communication systems or monitoring of laboratory experiments.

3- How is the language described in current documents?

Formal description (syntax notation), functional description stating the effect of typical instructions, definition of language subsets and extensions.

Is the language formally defined and in what form, e.g. BNF?

ALGOL 68

Yes, with the syntax specified as a two level van Wijngaarden grammar enabling mode equivalence and coercion (conversions) to be brought into the syntax. The semantics are described by algorithms written in stylised English.

CORAL 66

The syntax and semantics of the language are defined in the "Official Definition of CORAL 66" published by "Her Majesty's Stationary Office" using modified BNF for the syntax. This is not complete enough for a compiler unit to use as his only reference and a group of people are available at the Royal Radar Establishment for further help.

PAS 1 - PL/I

The syntax is defined in a BNF-like notation. The effect of instructions is given by detailed language descriptions.  
No subset or extension.



#### PEARL

"This language proposal has been worked out by a group of authors from several German industrial firms and research institutes. The report is intended to be a working document for the implementer rather than a tutorial paper or a handbook. so it was attempted to give a description which is as complete as possible as to the technical contents and results of the discussions and the authors took the risk of a joint paper such as might be inhomogeneities and less elegant presentation.

It might also be that there will be several modifications of contents and definition of items in the course of time when practical requirements will demand this."

The syntax notation used in this paper follows the extended Backus notation used to describe the concrete syntax of PL/I.

#### PL/I

The language is formally defined by the draft proposed international standard. The definition consists of four parts, the concrete syntax, the abstract syntax, the translator and the interpreter, using a defined metalanguage for each.

#### PROCOL

Description in BNF and in extended BNF exists. The description in VDL has been envisaged.

#### RTL/2

The syntax is defined in a form of BNF and the semantics by descriptions in English.

A subset is defined in which less secure features of the language are omitted. This subset is intended for application programs and the full language for system programs.



General contents

(In this section and the following, the given answers should not discuss tasking or data transfer features, which are relevant to separate questionnaires. They may refer to such features for completeness or because of interaction with the matter of this questionnaire.

4- What are the character sets defined by the language report?

ALGOL 68

No fixed character set defined. Terminals in syntax are things like 'begin-symbol', 'open-symbol'. Representations are given in the Report but the implementor is free to add representations of his own.

CORAL 66

Most actual implementations:

<u>Upper case letters</u>	A/B....X/Y/Z
<u>Lower case letters</u>	a/b....x/y/z
<u>Decimal digits</u>	ø/1/2/3/4/5/6/7/8/9
<u>Octal digits</u>	ø/1/2/3/4/5/6/7

Comparative operators = <= >= <> < >

Additive operators + or -

Multiplicative operators \* or /

Assignment operator : =

and: separator for array bounds and a terminator for label setting.

Statement terminator is ;

Brackets for nesting expressions or lists ( )

Brackets for array subscripts and table index are [ ]

String start and end delineators "....."

Parantheses for key-words '.....'

Separator ,

Point for numeric constants (decimal or octal)

Exponentiation to the base 1ø

CORAL 66 (continued)

Layout characters, such as space, carriage return and line feed, are ignored except where they occur in strings.

PAS 1 - PL/I

55 characters:      letters                      A....Z  
                     digits                      Ø....9  
                     special characters = + - \* / ( ) , . ' ;  
    7 & 1 > < ? bl (blank)

These 55 characters are contained in the 6Ø character-set of full PL/I. Programs may also be written with the 48 character-set of PL/I by replacing some special characters:

6Ø character-set:   ;   :   7   &   |   >   <   ?

48 character-set:   ..   .. NOT AND OR GT LT   ,

PEARL

The character set of PEARL is described by the following production-rules:

character : : = alphanumeric-character/special-character

alphanumeric-character : : = letter/digit

letter : : = A/B/ ..... /X/Y/Z

digit : : = Ø/1/2//4/5/6/7/8/9

special-character : : = lf / cr /    /' / ( / )    /\* /

                  + / , / - / . / // : / ; /    / = /

where lf means line feed, cr means carriage return, and    means space. The collocation sequence of the special-characters is implementation dependent.

The PEARL character set comprises 53 (at most 59) characters, so that a character may be represented by 6 bits if necessary.

PL/I

The language character set "is a finite set of symbols comprising 56 language characters, and zero or more extralingual-character~~s~~ which are...implementation defined".

PL/I (continued)

$\langle \text{symbol} \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z$   
 $\emptyset|1|2|3|4|5|6|7|8|9|.|\backslash|'|=|+|-|*|/|( )|,|_|$|;|:|\epsilon$   
 $|\langle \rangle| \langle \text{extralingual-character} \rangle$

PROCOL

ASCII

RTL/2

61 characters	letters	A B C ... Z
	digits	$\emptyset$ 1 2 ... 9
	layout	tab newline space
	other	" # £ \$ % & ' ( ) * + , - . / : ; < = > ? @

The characters # £ \$ are interchangeable. The characters & ? @ may only occur in strings, comments and titles (and possibly in machine code sequences).

5- What facilities exist in the language for data and program protection?

ALGOL 68

The compile-time checking of modes makes data reference between incompatible modes impossible. Array bound and ref scope violations would require run-time checks. Synchronization and data protection between parallel activities may be achieved using semaphores.

CORAL 66

Array bound checking and SWITCH and TABLE checks are often available at run-time.

The language permits independent compilation of segments and prohibits references which are external to this segment unless by explicit declaration.

The language is block structured thus offering some protection via restricting the scope of identifiers.

PAS 1 - PL/I

Semaphore-variables, optional checking of type conversion, overflow, zerodivide, endfile and transmit errors.

PEARL

Data protection by using the short-hand mechanism is possible (see 20). No explicit program protection facilities are provided because there is no necessity at language level. Some kind of program protection may be achieved using the synchronization operations and semaphore variables.

PL/I

The language is block structured. References are internal to the block unless specifically declared otherwise. Explicit declaration of names within a block override declaratives inherited from the containing block. Conditions are raised on various possible errors.

PROCOL

Program protection: by use of semaphores

Data protection: run-time checks (array bounds, switch statements etc.)

RTL/2

The array bound and stack overflow checks mandatory at run-time in the application language ensure almost complete data and program protection.

6- What kind of source language library reference may be made at the language level?

ALGOL 68

No methods are given within the language. Could be done using pragmats.



CORAL 66

Explicit reference to library elements within a segment are made under a library declaration. All library elements are procedures and all reference to these are procedure calls.

PAS 1 - PL/I

None.

PEARL

None.

PL/I

Many implementations provide a preprocessor or macro language facility of varying degrees of capability. Some allow conditional compilation of text included from a source language library.

PROCOL

None.

RTL/2

None.

Definition of data

7- What are the rules for interleaving declarations and executable statements?

ALGOL 68

Executable statements between declarations at the start of a serial clause cannot be labelled. Statements following the last declaration in a serial clause may be labelled. (A serial clause is a sequence of declarations or unitary clauses between begin and end. An assignment statement is a unitary clause.)



CORAL 66

Declarations must appear at the beginning of a block and must precede any executable statements in that block.

PAS 1 - PL/I

Declarations and executable statements may have an arbitrary sequence. They must be within a block.

PEARL

Within a block all declarations must precede the first executable statement.

PL/I

Declarations and executable statements may be mixed. The scope of a declaration is not affected by its position within a block. Declarations may be labelled and are treated as null statements at execution time.

PROCOL

Declarations must precede the procedure-body part of the task and/or the subroutine.

RTL/2

Declarations of local data must precede the statements of the block concerned.

Global data is declared in data bricks; data bricks and procedure bricks may be declared in any order.

8- How is the scope of an explicit or implicit declaration defined?

ALGOL 68

The 'usual' ALGOL block rules with extensions for if...fi and others.

#### CORAL 66

The scope of an identifier is restricted to the block containing its declaration whether implicit or explicit as in ALCOL 60, an identifier name may be re-used in an inner block.

#### PAS 1 - PL/I

Scope of variables is defined by attribute and/or block structure.

#### PEARL

Identifiers declared on problem-division level without a GLOBAL attribute are local to the module in which they are declared.

#### PL/I

"The scope of an explicit declaration of a name is that block to which the declaration is internal, including all contained blocks except those blocks to which another explicit declaration of the same identifier is internal."

"The scope of a contextual declaration of a name is determined as if the declaration were made in a DECLARE statement immediately following the PROCEDURE statement of the external procedure in which the name appears."

"The scope of an implicit declaration of a name is determined as if the name were declared in a DECLARE statement immediately following the PROCEDURE statement of the external procedure in which the name is used."

#### PROCOL

The scope of a variable is the procedure bearing its declaration. Variables declared in the COMMON unit are likely to be used by all tasks. Variables declared within a task (or a subroutine) are restricted for use within this task (or subroutine).

RTL/2

Local data (including literal labels) follows the normal ALGOL block rules.

Global data (data brick variables, procedures etc.) are in scope throughout the module.

9- Is it necessary to declare the affiliation connected to an identifier explicitly, or are there any implicit declarations?

ALGOL 68

No implicit declarations.

CORAL 66

No implicit declarations except for labels.

PAS 1 - PL/I

Binary variables of PAS 1-programs are implicitly declared.

PEARL

No implicit declarations.

PL/I

There are implicit declarations.

PROCOL

No implicit declarations. (\*)

RTL/2

No implicit declarations. (\*)

10- Is it necessary to declare identifiers before they are used?

ALGOL 68

All identifiers must be declared, although textually use may precede declaration to allow for mutual recursion.

CORAL 66

All identifiers must be declared before their use except labels which are declared at their point of occurrence.

PAS 1 - PL/I

Names are given to data by explicit declaration, exception:  
binal variables of PAS 1-programs.

Identifiers are not to be declared before they are used, exception: process-peripherals are named in the so called EQUIPMENT-description at the beginning of a PAS 1-program.

PEARL

All user defined identifiers must be declared before they are used except for labels.

PL/I

No.

PROCOL

Identifiers need not be declared before they are used.

RTL/2

Identifiers must be declared before use except in the case of labels and global identifiers - that is brick names and variables in data-bricks or identifiers denoting literal labels.

11- How are programmer defined identifiers introduced?

ALGOL 68

In declarations.



CORAL 66

By explicit declarations.

PAS 1 - PL/I

By explicit declaration.

Binary variables in PAS 1 are declared implicitly.

PEARL

To define the properties of various identifiers used in a PEARL program there are declarations necessary. Each programmer defined identifier must be introduced by an explicit declaration. The declare-sentence starts with the key-word

DECLARE or DCL.

PL/I

By explicit, contextual and implicit declarations.

PROCOL

By means of explicit declarations statements either in the COMMON or in the beginning of tasks and (external or internal) procedures.

RTL/2

By explicit declaration.

12- What restrictions are there on the programmer in the free choice of identifiers?

ALGOL 68

None.

CORAL 66

None.



PAS 1 - PL/I

No restriction but in actual implementation the number of significant characters is restricted to 6.

PEARL

There are no restrictions given in the language report.

PL/I

None in the language.

PROCOL

No use of key-words (and the set of forbidden words used by the assembler of particular machine of implementation).

RTL/2

A name which denotes a key-word cannot be used as an identifier.

13- Does the declare statement start with a specific key-word?

ALGOL 68

With specific strop (underlined) word giving its mode e.g.

real x;.

CORAL 66

The declaration statement starts with a key-word from a set of key-words which specify the mode of the items declared.

E.g. 'FLOATING' X; 'INTEGER' Y;.

PAS 1 - PL/I

Declarations are introduced by the key-words DCL or DECLARE.

PEARL

The declare-sentence starts with the key-word DECLARE or DCL. There are other declarative statements starting with the key-words TYPE or OPERATOR or OPTR, also with TASK or TASKNAME!

PEARL (continued)

Not all identifiers used in I/O-statements are introduced in declare-sentences (see description of I/O-features).

PL/I

Yes, DECLARE or DCL.

PROCOL

Yes, and the key-word indicates the type of the data (except for P-structure and item): INTEGER, SHORTEGER, REAL, BINARY, STRUCTURE, ITEM.

RTL/2

Yes, with a key-word or group of key-words specifying the mode of the items being declared.

E.g. REAL X;

REF ARRAY BYTE T;.

14- Is it possible to group together in a declare statement identifiers with the same attributes (type, mode)?

ALGOL 68

Yes, e.g.: real x; real y; real z; may be written  
real x,y,z;.

CORAL 66

Yes, a list of identifiers may follow the key-word in a declare statement.

PAS 1 - PL/I

Yes, by enclosing a list of identifiers in brackets.

PEARL

Yes, it is by replacing the single identifier by a list enclosed in brackets.

AD-A032 571

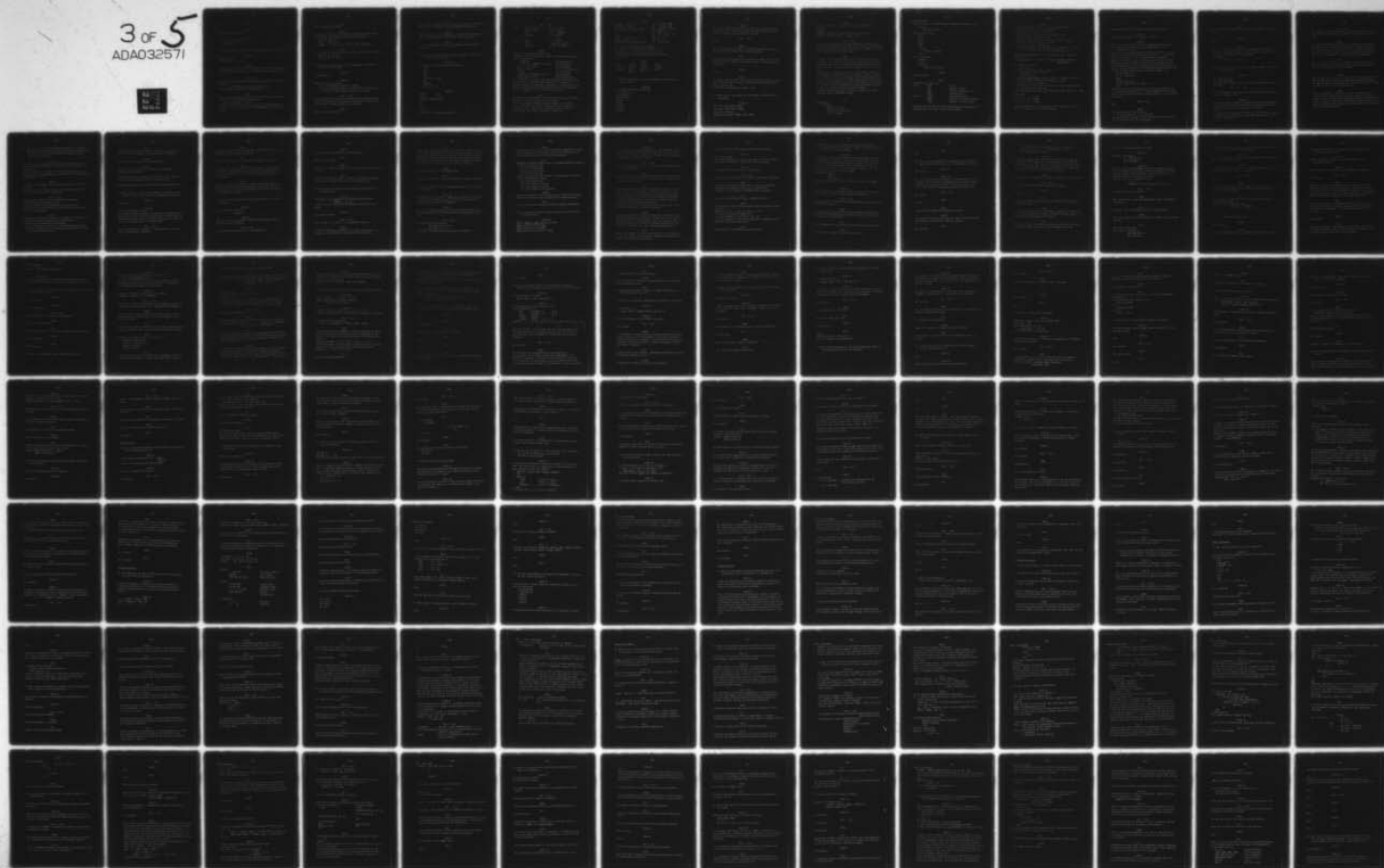
PURDUE UNIV LAFAYETTE IND PURDUE LAB FOR APPLIED IND--ETC F/G 9/2  
A LANGUAGE COMPARISON. A COMPARISON OF THE PROPERTIES OF THE PR--ETC(U)  
OCT 76 R ROESSLER, K SCHENK

N00014-76-C-0732

UNCLASSIFIED

NL

3 of 5  
ADA032571



PL/I

Yes.

PROCOL

Yes, separated by commas.

RTL/2

Yes, separated by commas, e.g.

REAL X,Y,Z;.

15- How are declared variables initialized with specific values?

ALGOL 68

real x: = 4.0;  
[1:3] int i: = (1,6,-48);.

CORAL 66

Variables can be initialized at compilation by presetting within the declaration statement or at run-time by the assignment statement.

PAS 1 - PL/I

Using the INITIAL-attribute within the declaration of PL/I-variables

DCL X BIN FIXED (15) STATIC INITIAL (123);

Events in PAS 1-programs may be initialized.

PEARL

They are initialized using the initial-attribute within the declaration. As an example the declaration

DCL (I,J,K) INT: = (3,5,7);

declares the identifier I,J,K in a way as if the assignments I: =3; J: =5; K: =7; were executed immediately after the declaration.

PL/I

With the INITIAL attribute.

PROCOL

All types of data may be initialized using the key-word 'INIT' (except formal parameters of a subroutine). Example:

```
INTEGER A INIT 123;  
REAL    B INIT 36.5;  
BINARY  C (3) INIT (Ø' 37', Ø'36', Ø'35') LENGTH 6;.
```

RTL/2

By following the identifier by := and the initial value. Example:

```
REAL X:=6.9, Y:=3.2;  
ARRAY (3)INT I:=(2,4,6);.
```

16- Are there restrictions - or is it possible to initialize each kind of variables?

ALGOL 68

No restrictions.

CORAL 66

Presetting is only allowed

- (i) in the outermost block of a segment
- (ii) in the outermost block of a non-recursive procedure declared in a block where presetting is allowed.

PAS 1 - PL/I

Only variables of storage class STATIC may be initialized that means initialization is performed at compile-time. Binal variables of PAS 1-programs cannot be initialized.

PEARL

There are no restrictions.



PL/I

The INITIAL attribute cannot be given to constants, DEFINED data, structures or parameters (except CONTROLLED parameters).

PROCOL

Restrictions: it is not possible to initialize variables which are overlayed on other variables (by a LIKE statement).

RTL/2

It is not possible to initialize variables of mode LABEL in a data brick.

17- How are constants written? (Give an example or definition of each kind of notation).

ALGOL 68

Constants are called denotations. Examples:

3.14

true

"a"

4096

1.23E4 or 1.24<sub>10</sub><sup>4</sup>

1011

"string"

\$pt, 3(d.d) 1\$

CORAL 66

'OCTAL' (37)

'OCTAL' (37.32)

"THIS IS A STRING 123"

4.1

+4.1

-4.1

3.89<sub>10</sub><sup>-4</sup>

LITERAL (X) / Printing character X

# PAS 1 - PL/I

PL/I :	binary fixed	2/ 107 / 101B
	binary float	2.5 / .9EO3
	bit	'1'B / '10010'B
	character	'C' / ABC05'
PAS 1 :	binal	317 / 10011B
	clock	12.30 (hour, minute)
	duration	12 MIN / 25 MSEC

## PEARL

There are constant-denotations for:

arithmetic-, binal-, string-, format-, clock and duration-constants. Examples are:

1, 1000,007	integer-constants
.5, 0.5, 7E5, .3000E-10, 59.379E3	real-constants
B'0', B'1', B'1010'	binal-constants
0'537', H'D8F1'	-"- (octal hexadecimal)
'ABCDEF'	string-constant
18:10:30.0 (10min30sec past 18)	clock-constant
2 HRS 3 MIN 5.5 SEC	duration-constant

It is possible to use as string-characters non-PEARL-characters not contained in the PEARL character set. "if contained in a string, denotes one apostrophe, otherwise it is the empty string. Format-constants look like string-constants but are preceded by an F. Example: F' (3) (I (7), I(17)), R(6,6,6)' .

## PL/I

Arithmetic data have base, scale, precision and mode attributes. String data can be character or bit strings.

3.1416 is a decimal fixed point real constant of precision (5,4). -10.01E99B is a binary floating point real constant of precision (2,0). 'PAGE 5' is a character string constant. (64) '0'B is a bit string constant of length 64.

# PROCOL

INTEGER ( 262 143 )	Ex: $\emptyset / -137\emptyset / -16\emptyset\emptyset\emptyset$
SHORTEGER ( 2 $\emptyset$ 47 )	Ex: $1 / +4 / 2\emptyset 47 / -2\emptyset 47$
The same with a scale factor.	Ex: $175\emptyset .7 \text{ B}3 / -1\emptyset .2 \text{ B}1$
	Ex: $2\emptyset\emptyset .7 \text{ B}3$
REAL	Ex: 1 2 3 4 5 6 . 7891
OCTAL ( 1777777 )	Ex: $\emptyset ' 1777 777 '$
OCTAL - SHORT ( 7777 )	Ex: $\emptyset ' 7777 '$
CHARACTER (initialization only)	Ex: C' AB + bb'

For array: (in a declaration statement)

Ex: BINARY T (3) INIT ( $\emptyset '77', \emptyset '6671', \emptyset '1'$ );

INTEGER B (8) INIT (1 $\emptyset$ , 2 $\emptyset$ ,  $\emptyset$ , 4, 6, - 3, 12);

No overlapping possible with the overlay statement:

LIKE (cf 52).

## RTL/2

Real	3.47	$\emptyset .1\text{E}7$	76E-4
Fraction	$\emptyset .493\text{B}1$	17.6B-5	1E1 $\emptyset$ B-36
Integer	999	BIN 1 $\emptyset$ 1	HEX F7
	OCT 77	'A'	

18- What conversions, if any, can be applied to constants at compile time?

## ALGOL 68

All possible language conversions

(coercions) viz

deproceduring

dereferencing

uniting

rowing

widening

hipping

voiding .

CORAL 66

The "Official Definition" does not state how a constant should be held in store. This does not preclude conversions being performed at compile time according to context.

PAS 1 - PL/I

None.

PEARL

Not necessary, because there is a constant notation for every kind of data to be handled by a PEARL statement.

PL/I

All necessary conversions may be applied at compile time as the language does not specify the internal format in which constants are stored.

PROCOL

None.

RTL/2

An integer constant whose value lies in  $[-0,255]$  is taken, a priori, as of mode BYTE and in certain contexts may be widened to mode INT at compile time.

Otherwise no conversions at compile time.

19- What kinds of data types can be handled, and how are they described?

ALGOL 68

Primitive data types (modes):

int, real, char, bool, format

others by extension:

string, bits, bytes, compl, file, sema.



CORAL 66

FIXED (n, m) where n = width and m = scale (precision)

FLOATING

INTEGER

TABLES as described at question 40 are also allowed

ARRAYS of FIXED, FLOATING or INTEGER items.

PAS 1 - PL/I

Binary fixed and binary float with choosable precision, bit- and characterstring with choosable length, event- and semaphore-variables, different storage classes: static, automatic, arrays of all types of data mentioned above.

PEARL

In PEARL one can handle integer and real data with unlimited precision. For a specific variable the programmer has to define a specific precision (or use the implementation defined precision by default). The key-words for arithmetic data are INT and REAL.

In PEARL one can handle bit-pattern with arbitrary length. For each variable of mode BIN one has to choose a specific number of bitpositions or use the implementation defined length by default. It is also possible to handle character strings of arbitrary length. Variables with the attribute STRING are of programmer defined length or implementation defined length. The same is true for variables with the attribute FORMAT.

Variables may be connected also with attributes like CLOCK, DURATION, DEVICE, INTERRUPT, SIGNAL.

PL/I

Problem Data

Arithmetic

decimal or binary

fixed or floating point

real or complex



PL/I (continued)

NOTE: precision is a specificable attribute of arithmetic data.

String

character or bit

Program Control Data

Label

File

Entry

Event

Task

Locator

pointer or offset

Area

Data Organization

Elements

Arrays

Structures

Arrays of Structures

PROCOL

See next page.

RTL/2

Primitive data types are:

plain:	BYTE	8 bits
	INT	integer (signed)
	FRAC	fraction in range (-1,1)
	REAL	floating point
control:	LABEL	level/address pair
	PROC	pointer to literal procedure
	STACK	pointer to literal stack

Note that INT and FRAC will have range/precision defined by the implementation. At least 16 bits can be assumed.

PROCOL

- Isolated data whose length is one or two words long;
- arrays of such data (1 or 2 dimensions)
- P-structure (isolated) (See 37.)
- items composing a P-structure (less than 1 word)
- arrays of P-structure (1 or 2 dim)
- arrays of 'simple object' (less than 1 word in length)  
the simple object is in fact an item.
- arrays of structure: dimensions  
Ex: STRUCTURE TSTRU (10, 20) ITEM (1,0) UN ITEM (18, 1) DEUX;  
UN (I, J) designates the item UN in TSTRU (I, J)

Description of identifier (1 to 6 letters or digit, the first is a letter)

- isolated word (1 or 2 words): its type: INTEGER, SHORTEGER,  
REAL, BINARY

Ex: INTEGER VAL, MUL;

REAL LAR, RU1;

- arrays: its dimensions and its type

Ex: INTEGER VAL (10), MUL(3,2);

REAL LAR (5,5);

- arrays of simple object: Type is 'binary'; dimensions of the arrays and length of the simple objects (in bits)

Ex: BINARY TOS (36) LENGTH 19;

- isolated P-structure: type is "Structure"; the composing items are declared at the same time

Ex: STRUCTURE S ITEM (4,0) ALFA ITEM (10,2) BETA ITEM (7,12) GAMMA;

Bit-numbers:

0, 1, 2, 3             $\hat{=}$  ALFA

2, 3, 4, 5, ..., 11  $\hat{=}$  BETA

12, 13, ..., 18      $\hat{=}$  GAMMA

Items are specified by their length in bits and the bit of origin in the structure (from left to right).

20- What short-hand for constant notations is available?

ALGOL 68

Identity declarations, e.g. real pi = 3.14159;

CORAL 66

None in the language, but one could always set up macros for these and use as constants in a parametric manner, i.e.

```
'DEFINE' PI "3.14....."
```

would in a global manner replace each occurrence of PI in the text of the program by the numeric value 3.14..... as the program is read (but not if PI occurred within a comment).

Note that the above use of the CORAL macro features is a very simple application as CORAL macros can have parameters which may themselves be other macro names and could be indirectly recursive. This constitutes a very much more powerful feature than the use of CORAL macros given in the above example, i.e.

```
'DEFINE' FRED (a)"
```

```
N: = a-1; BILL (N)"
```

```
'END';
```

```
'DEFINE' BILL (b)"
```

```
'IF'N=b 'THEN' FRED (b) 'ELSE'....."
```

```
'END';
```

These features of the language give a much more powerful form of a macro processor which is a language element in CORAL than the use one may suggest from the above example of PI for setting a constant.

PAS 1 - PL/I

None.

PEARL

For example, using the declaration

```
DCL PI VAL REAL (11) = 3.14159 26535
```

the identifier PI becomes a short-hand and may never be used as the left-hand part of an assignment.

PL/I

Using DECLARE statements and the INITIAL attribute or by assignment.

PROCOL

None.

RTL/2

There is a simple replacement facility whereby any series of items not containing a semicolon can be given a name. This is primarily intended for naming constants.

For example:

LET PI = 3.14159;

21- Are data items of different bit length allowed, and how are they manipulated?

ALGOL 68

Yes - by using long

e.g. long long bits

would be three words as a bit string. Manipulated by operators provided:

$\vee$ ,  $\wedge$  etc.

It is impossible to declare a bit string that will use a fraction of a computer word.

CORAL 66

Yes, fixed point arithmetic variables have their length specified in bits. The official definition suggests using the next larger computer word to contain such a variable.

Also:

An operator is available which extracts , at run-time, a specified field from a declared variable and manipulates it as an integer (ref see 27).



PAS 1 - PL/I

Data items may have different bit length. The precisions of different items are automatically adjusted in operations and assignments.

PEARL

Data items of different bit length are allowed. By introducing an identifier for a specific variable the programmer can determine the precision in decimal digits for arithmetic variables, the length in bit positions for bit patterns and the length as the number of characters for string- and format-variables.

PL/I

Yes, bit string variables and constants are allowed. They can be manipulated by the string operations and built-in-functions and can be converted to other data types.

PROCOL

Yes, an item is a fraction of a word that has been declared with its own identifier; also some simple objects may be less than one machine word in length. The first bit position and the length of item are specified in the declaration.

RTL/2

Strictly speaking, no. Bytes can be used for small integers ( $< 8$  bits) but otherwise the unique word length is machine dependant. Double length forms automatically arise as intermediate values in certain fixed point operations but they cannot be stored in variables. Shift operations can be applied to these double forms for scaling purposes.



22- What are the different precisions available for arithmetic variables, and are these different precisions implementation defined, or is it possible to choose a minimum precision within the declaration?

#### ALGOL 68

Precisions are implementation defined. Environment enquiries are provided for determining the precisions in a particular implementation.

Default chosen by e.g. real x, smaller precisions may be chosen by short real x, short short real x and longer ones by long real x, long long real x etc.

#### CORAL 66

'INTEGER' and 'FLOATING' have precisions defined by the implementation.

'FIXED' (n, m) for FIXED variables, the programmer specifies his own length and decimal precisions.

#### PAS 1 - PL/I

Limitations in precision are implementation dependent:

Bit data may have 1 through 32 bits

Binary fixed data may have precision 1 through 30 bits

Binary float data have precision 23 bits for the mantissa and 8 bits for the exponent.

#### PEARL

Variables with different precision or length are treated as variables with different mode with all implications.

- Precisions are implementation dependent
- Any precision may be specified. It is expected that the compiler will choose the next implementation dependent precision to realize the required one.

PL/I

Precision of an arithmetic datum is a declarable attribute of the datum. Any limitations are implementation limitations.

PROCOL

The precision is fixed by the implementation.

RTL/2

No choice of precision is available to the programmer.  
BYTES are always 8 bits.

INTs and FRACs have range/precision defined by the implementation but all implementations have 16 bits or more.

REALs have precision and range defined by the implementation.

23- When there is a data type "bit-string" available, is the length of such a bit-string only implementation defined?

ALGOL 68

Yes, see 21.

CORAL 66

No, "bit-string" but see 27.

Any INTEGER, FIXED or FLOATING variable may be interpreted as a bit-string and may be used as an operand of a DIFFER, UNION or MASK operation. The result of such an operation is a bit-string that can be interpreted as an INTEGER value.

PAS 1 - PL/I

Bit-strings may have a length of 1 through 32 bits. The maximum is implementation dependent.

PEARL

A variable of mode BIN has a programmer defined length or a implementation defined length by default.

PL/I

No, bit-string length is a declarable attribute of the datum.

PROCOL

In PROCOL, the maximum length of an item and of a 'simple object' is limited to the length of the word of the implementation. But, there is the possibility to use arrays of simple objects and of structure.

RTL/2

There is no mode 'bit-string' as such but the integer mode may be used as a bit-string and the representation is defined to be 2s complement so that correspondence is explicit. The length is implementation defined.

24- What are the restrictions concerning the length of a string variable?

ALGOL 68

None - excepting store size.

string x;

CORAL 66

There are no string variables in CORAL, only integers that may take the value of a string.

PAS 1 - PL/I

Character-strings may have up to 128 characters.

PEARL

The length must be defined in the declaration.

PL/I

None in the language.

PROCOL

There is no string variable, only a string constant for initialization.

RTL/2

There is no string variable: A string is a constant denotation of an array of bytes; there are no size restrictions.

25- What are the variable formats available and how can they be manipulated?

ALGOL 68

Variables of mode format can be declared and initialized

format x: = \$...\$

formats can only be assigned and passed to and returned from procs.

CORAL 66

No variable formats.

PAS 1 - PL/I

Only in LINE (n)-, SKIP (n) - and COLUMN (n)-option.

PEARL

A format in PEARL is comparable to a string. Operations on variables of type FORMAT are concatenation and assignment.

PL/I

There is no format data type per se in PL/I but there is a remote format item which can readily be used to simulate such data types when required. PL/I has list and data directed input/output as well as edit directed. This last uses format statements but the first two allow free format input, therefore, reducing the need for a format data type or simulation thereof.

PROCOL

No variable formats in the T2000 version.

RTL/2

There are no variable formats in the sense of special variables.

26- How is calendar and clock information handled, and what are the units?

ALGOL 68

Not defined in the language - procs may be provided in standard prelude. (The standard prelude is part of the outer block in which the user's program is embedded).

CORAL 66

Not in the language, but could be made available through standard procedures or more effectively by macros producing inline code.

PAS 1 - PL/I

By special built in functions

CALL TIME (H, M, S, MS):

CALL DATE (D, M, Y);

Additional clock and duration variables are possible.



PEARL

There are two kinds of data modes, CLOCK and DURATION. Variables of each mode are to be declared. There is the possibility to build clock-expressions and duration-expressions.

PL/I

The built-in-function DATE returns a character string of length 6, in the form yymmdd, where:

yy is the current year

mm is the current month

dd is the current day.

The built-in-function TIME returns a character set of length 9, in the forms hhmmssstt, where:

hh is the current hour

mm is the number of minutes

ss is the number of seconds

tt is the number of milliseconds.

PROCOL

Calendar information is not handled, only relative time (represented by integers) period is specified at generation time.

RTL/2

Not in the language. Normally handled by standard procedures.

27- Can computer words be divided into smaller units and how are they handled?

ALGOL 68

Bits or bytes or short int can be used.

Bytes divides word into char.

Bits divides word into bool.

Short int uses half word as integer.

CORAL 66

Various fields of variables declared as type 'INTEGER' 'FIXED' and 'FLOATING' can be accessed by using the 'BITS' operator the result of which is handled as an integer. Part word elements are possible in 'TABLE' (see 4Ø).

PAS 1 - PL/I

Yes, by packing several items of arrays into one computer-word (attribute UNALIGNED).

PEARL

The language is not concerned with the internal representation of any data. Therefore no explicit definition of parts of computer-words are sensible.

PL/I

The language does not define a relationship between its data types and the concrete machine on which it is being executed. It does require that the mapping be discoverable (the UNSPEC built-in-function) and has attributes, ALIGNED and UNALIGNED, to guide an implementation to the priority between speed of access and storage economy respectively. Conversion can be easily effected among the data types. Structures of bit strings can easily be declared.

PROCOL

The language provides a means to handle bits and fractions of words. The language can manipulate 'items' and 'simple objects' which are less than one word long. They have names and they are handled as logical or arithmetic variables (depending on the context in which they are used). They may be subscripted.

RTL/2

Not in the language. In practice the BYTE may be thought of as a part word. Shift and other logical operators enable parts of words to be extracted and updated.

28- Can data items share storage with other data items?

ALGOL 68

Yes, using unions.

For example: union rib = (real, int, bits); Run-time information must be carried to indicate current mode of rib.

CORAL 66

Yes, 'OVERLAY' similar to Fortran equivalence.

PAS 1 - PL/I

No equivalence declaration of different data items is possible.

PEARL

Variables with different precision or length are treated as variables with different mode with all implications.

No mapping in the FORTRAN EQUIVALENCE sense is possible.

PL/I

Yes, using the DEFINED attribute or BASED variables.

PROCOL

The instruction LIKE allows the definition of any data by means of another.

Example: ----- D (1Ø, 1Ø) LIKE A -----

D (1Ø, 1Ø) has the same address as A.

It is not possible to initialize D (1Ø, 1Ø).

It is not possible to use LIKE between data in COMMON and data internal to task and subroutines.

RTL/2

Not possible in the FORTRAN equivalence sense.

29- What facilities are provided for handling packed data?

Can substrings of a bit-string be named, referenced and operated upon as for bit-strings?

#### ALGOL 68

In ALGOL 68 a bit-string is declared using mode bits. (An item of mode bits is a single storage unit in length (say 32 bits), an item of mode long bits would be two storage units (64 bits) and so on.) The only substring that is possible to refer to within such an object is the single bit as a bool, such a bit being obtained using the operator elem.

For example: bits p;

bool b;

b:=6elem p;

would assign true to b if the sixth bit of p was 1 and false if 0.

#### CORAL 66

Within a table it is possible to refer to parts of a bit-string by overlaying it with a different table declaration.

#### PAS 1 - PL/I

Substrings can be referenced and operated upon as for bit-strings but cannot be named.

#### PEARL

The language is not concerned with the hardware representation of data. The implementation may provide packing of e.g. arrays or structures of bit-strings.

#### PL/I

Yes, by SUBSTR built in function and by string overlay defining.

#### PROCOL

Yes, these bit-strings are called item (see 19).

RTL/2

None.

30- Can a certain juxtaposition of substrings be ensured and what limitations exist arising from hardware boundaries?

ALGOL 68

Not relevant.

CORAL 66

Yes, no item may cross a boundary of a word as defined in the language. This can cause some problems on byte machines where the CORAL 66 word may be 16 bits. To allow an item to then cross a byte boundary sometimes produces inefficient code.

PAS 1 - PL/I

Not relevant.

PEARL

See 29.

PL/I

Yes, by using ALIGNED and UNALIGNED attributes.

PROCOL

Yes; items may overlay each other but cannot cross the machine word boundary (for efficient handling reasons).

RTL/2

Not relevant.



31- Can a whole string or substring of bits be treated as a different data type (e.g. arithmetic or logical)?  
What restrictions are there?

ALGOL 68

An item of mode bits can be converted to an item of mode int using the operator abs, the reverse operation being possible using the operator bin. There is an automatic conversion (coercion) between bits and `[ ] bool` enabling an item of mode bits to be treated as an array of boolean quantities.

CORAL 66

A bit-string can always be used as an arithmetic data type (see 23).

PAS 1 - PL/I

Yes, by using a conversion operator (e.g. FIXED).

PEARL

See 29.

PL/I

Yes, in the sense that bit-strings may readily be converted to other data types.

PROCOL

Yes; this can be treated as an integer, shorteger (arithmetical or logical) inbedded in an item or a structure whose length is restricted to the machine word length.

RTL/2

The only forms of bit-strings in RTL/2 are integers (and bytes). Integers in RTL/2 are held in two's complement notation.

32- How are address variables declared?

ALGOL 68

Examples: ref real x;  
          [ 1 : 10 ] ref int y;  
          ref [ ] compl z;

CORAL 66

They are declared as INTEGERS and used as address variables by the LOCATION facility which returns the address of an item and the anonymous reference which allows the contents of an INTEGER variable to be used as an address.

Examples are: i: = 'LOCATION' (var);  
              [i] : = 1;  
              'COMMENT' would set contents of var to 1;

PAS 1 - PL/I

No address variable available.

PEARL

They are declared in the same manner that normal variables are introduced.

PL/I

By usage, that is, statement prefix, or by declaration.

PROCOL

There is no address variable available in PROCOL (in the current version).

RTL/2

Using the key-word REF.

For example: REF REAL X;  
              REF ARRAY BYTE S;  
              REF LIST CELL;

33- Can one pointer only point to data of one specific type and how is this expressed in the language?

ALGOL 68

Yes. (All checkable at compile time.) Association with data item made in declaration. See 32.

CORAL 66

No check is made on the mode of a variable pointed to by an integer that is being used as an address.

PAS 1 - PL/I

Not relevant.

PEARL

Pointers are mode restricted. If REAL is the attribute of a variable then REF REAL is the attribute of a pointer able to point to a real variable.

PL/I

No, a pointer variable can be used to identify locations of other based variables.

Example: DCL X CHAR (20) BASED (P),  
          Y (20) CHAR (1) BASED (P);

.

.

.

READ FILE (IN) SET (P);

PROCOL

Not relevant.

RTL/2

Yes, a variable declared as REF REAL can only point to a REAL.

34- Can we have pointers to pointers, and if so, what are the rules governing their use?

ALGOL 68

Yes, but multiple indirection is implicit in mode, e.g.

ref ref ref real x;

CORAL 66

Yes, no restriction on the use of an INTEGER as an address.

PAS 1 - PL/I

Not relevant.

PEARL

There are no pointers to pointers in the sense that a single pointer may refer to a place containing a single other pointer to an arbitrary place. But hierarchies of pointers are available by using pointers as elements of data structures or aggregates and by manipulating a pointer to such a data aggregate.

PL/I

Yes, "only the first or left most (locator qualifier) can be a function reference; all other locator qualifiers must themselves be based variables". "...they are evaluated from left to right".

PROCOL

Not relevant.

RTL/2

No. That is REF REF is not allowed. In practise a record mode can be defined with a single component being of mode REF...and a pointer to this record may then be declared.

RTL/2 (continued)

Example: MODE REFINT (REF INT RI);

.

.

REF REFINT X;

X would then effectively be a REF REF variable but component selection would be required to access the ultimate INT. (X.RI).

35- What are the restrictions concerning the number of subscripts (dimensions) of arrays?

ALGOL 68

No restrictions.

CORAL 66

Only 2 dimensions.

PAS 1 - PL/I

Only 2 dimensions. (Implementation dependent).

PEARL

There are no restrictions.

PL/I

None in the language. 15 is a common implementation restriction.

PROCOL

Only 3 dimensions.

RTL/2

No restrictions.

36- What are the possible types and modes for arrays?



#### ALGOL 68

All modes (types) may be declared as arrays, e.g.:

[1:2, -403:602] sema p; etc.

There is also a facility for declaring any bound of an array as flex which means that that bound may vary on assignment.

(Also either meaning flex or fixed bounds).

#### CORAL 66

Arrays of all types of simple data are allowed.

'INTEGER', 'FIXED' (n, m), 'FLOATING'.

#### PAS 1 - PL/I

Arrays of all types of data except arrays themselves may be declared. The lower bound must be 1. (Implementation dependent.)

#### PEARL

It is possible to build arrays of all kinds of data or data aggregates, other than arrays. So it is not possible to have arrays of arrays.

#### PL/I

All data types may be declared as arrays. Also, arrays of structures are allowed. Also, cross sections of arrays are possible.

#### PROCOL

Arrays of all kinds of data are allowed, i.e.

- arrays of simple object
- arrays of 2 words object
- arrays of structure
- arrays of "item".

#### RTL/2

Arrays can be of any mode except arrays themselves. Arrays of REF arrays are, however, allowed. The lower bound is always 1.

37- What kind of constant aggregates can be written?

#### ALGOL 68

The contents of structures and arrays may be written as constant aggregates. An example: mode title := struct (int date, string name);  
title book := (1066, "and all that");  
[1:3] int i := (1, 2, 3);

#### CORAL 66

Constant lists.

Example: ('OCTAL' (123), 256, 102) and

example: 3,4, -6.

A constant list may only be used as a list of preset values; it does not constitute a constant aggregate that can be used, for example, to assign a set of values to an array or table.

#### PAS 1 - PL/I

Only arrays of dimension 1 or 2. (\*)

#### PEARL

Constant aggregates denote either array-displays or structure-displays. Examples: (6,7,6) Vector with 3 components  
((3,-5), (7,9)) 2x2 Matrix.

#### PL/I

In PL/I, a constant aggregate is an aggregate with the INITIAL attribute which never appears on the left hand side of an assignment statement. There are few restrictions on the use of the INITIAL attribute.

#### PROCOL

In PROCOL, structure means a data contained in one word (called P-structure from now on) composed of (possibly overlapping) items. P-structure are of binary type and restricted to logical operations. 'Simple object' is a data less than 1 word long. (\*)

RTL/2

The contents of arrays and records may be written as a list of constants in brackets. In the special case of an array of bytes a string may also be used.

Example: (1,2,3) ((1.Ø,Ø.Ø), (Ø,Ø,1.Ø)) "STRING".

38- How are the initializations of arrays or structures written?

ALGOL 68

```
[1:2,1:2] int a := ((11,12), (21,22));  
struct (int year, [1:12] char comment) w  
      := (1Ø66, "and all that");
```

CORAL 66

```
'INTEGER' 'ARRAY' K1 [1:2,1:2] := 22, 11, 23, 21;  
'TABLE' can be similarly initialized with a constant list see 4Ø.
```

PAS 1 - PL/I

Initialization by declaration

```
DCL A (2) CHARACTER (3) INITIAL ('ABC', 'DEF');
```

PEARL

Array displays may be operands of operators declared for appropriate arrays, may be right-hand side of assignments and may occur in equivalent- and initial-attributes of identity-declarations.

Example: DCL ARRAY (1:2,1:3) INT: = ((2,3,4), (3,4,5));

Structure-displays only may be the right-hand side of an assignment or may occur in equivalent- and initial-attributes.

Example: (5.7, B'1Ø1', 115, 'ABC').

PL/I

Using the INITIAL ATTRIBUTE.

PROCOL

The INIT statement is exclusive of the LIKE statement; with respect to this, arrays are initialized as any other data:

A (5) INIT (-10, -9, -P, -7, -6);

Structures are P-structures in PROCOL (i.e. 1 word) and are initialized in the same way.

RTL/2

Initial values are denoted by following the identifier of the array or record by := and a list of initial values for the elements or components.

Example: ARRAY(2,2)INT AI:=((11,12),(21,22));

ARRAY (7)BYTE B:= "ABCDEFGH";

39- Is it possible to handle as one item, data collections or structures, consisting of components of different types?

ALGOL 68

Yes.

CORAL 66

Yes, 'TABLE'.

PAS 1 - PL/I

Questions not relevant for PAS 1 - BBC-PL/I

PEARL

Yes.

PL/I

Yes.

PROCOL

A P-structure is a word in which 'items', possibly overlapping, may be defined.

RTL/2

Yes, records.

40- How are the descriptions of these structures introduced  
(node by node like ALGOL 68, or full tree like COBOL records)?

ALGOL 68

By declarations of the form

```
struct (real i, int j, ref int p) r;
```

CORAL 66

By declarations of the following nature:

'TABLE' PERSON [3, 12]

[SEX 'UNSIGNED' (1) Ø, 8;

AGE 'UNSIGNED' (8) Ø, Ø;

HEIGHT 'INTEGER' 1;

WEIGHT 'FIXED' (16, 4) 2

'PRESET' (1, 23, 72, 156), (Ø, 21, 63, 112), (Ø, 28, 65, 126), (etc.),  
( ), ( ), ( ),  
( ), ( ), ( )]

This gives twelve 3 word records. The first word is broken into two fields which can be referenced by name. The next two words are used whole. Note the method of initializing the structure (optional).

PAS 1 - PL/I

Not relevant.

PEARL

The description is introduced within the declaration.

As an example: DCL RS STRUCT (VALUE REAL, NAME STRING (12));  
specifies a structure (a variable of type structure). RS refers to two elements, one whose mode is real and another whose mode is string and which is capable of receiving up to twelve characters.



PL/I

By declarations of the structure tree.

PROCOL

By one declaration statement, specifying both the P-structure and the composing items.

RTL/2

By describing the record template in a MODE definition e.g.:  
MODE COMPLEX (REAL RL, IM);

41- What are the possible embeddings of structures in structure?

ALGOL 68

To any depth, e.g.:

struct (real x, struct (bool d, int i)p) r;

CORAL 66

It is not possible to nest 'TABLE' structures.

PAS 1 - PL/I

Not relevant.

PEARL

It is possible to build arrays of structures and to build structures of arrays. It is also possible to put into a structure another structure as an element. It is also possible to put into a specific structure a pointer to another incarnation of the same type of structure.

PL/I

No restriction in the language. Implementations normally restrict the depth of nesting to, say 15.

PROCOL

No embedding possible: only array of P-structures.

RTL/2

It is not possible to embed a record within a record. However, it is possible for a component of a record to be an array and to have arrays of records.

42- What are the possible types or modes of structure elements (items, fields)?

ALGOL 68

Any mode can be used.

CORAL 66

'TABLE' elements can be any single word and part word variables and can have all three modes, 'INTEGER', 'FIXED' (n, m) and 'FLOATING'.

PAS 1 - PL/I

Not relevant.

PEARL

The elements of a structure may be of any type available.

PL/I

Any data type.

PROCOL

May be of any type (signed or unsigned). (\*)

RTL/2

Any, other than records themselves.

43- How are pointer variables placed as components into such structures?

ALGOL 68

In any position as in

struct (real i, int j, ref int p) r;

CORAL 66

There is no explicit type pointer in CORAL 66. However, facilities exist in CORAL 66 to allow the use of integers as addresses. Initialization of such integers must be programmed.

PAS 1 - PL/I

Not relevant.

PEARL

In the same way as other data.

PL/I

As for any other data type.

PROCOL

Not relevant.

RTL/2

This:

MODE CELL (INT HD, REF CELL TL);

TL is a component of mode REF CELL.

44- How are the references to the places where data items are stored manipulated within the language?

ALGOL 68

By assignment. By comparison both between themselves (using is and isnt) and with nil (an invalid address). They only differ from more conventional items in that only a few operations are defined for them.

CORAL 66

The address of any variable or structure can be assigned to an 'INTEGER'. Subsequently it is manipulated as an 'INTEGER'.

PAS 1 - PL/I

Not relevant.

PEARL

One can identify and store such references, one can compare two different reference-variables.

PL/I

By assignment and comparison.

PROCOL

There are no pointer variables within the language.

RTL/2

By assignment in the usual way and by comparison using `:=` and `:#`.

45- In which kinds of data aggregates may a pointer appear (arrays, structures, etc.)?

ALGOL 68

Any.

CORAL 66

Again, pointers do not exist as special data types.

PAS 1 - PL/I

Not relevant.

PEARL

Pointers may appear in each kind of data aggregates.

PL/I

Any.

PROCOL

Not relevant.

RTL/2

Any.

46- How is a linked list constructed?

ALGOL 68

```
mode cell = struct (int item, ref cell next);  
ref cell x: =nil;  
for i from 4 by -1 to 1 do  
    x: =heap cell: = (i,x) od;  
List constructed in reverse order.
```

CORAL 66

A linked list can only be created by programmed use of 'INTEGER' variables as addresses.

PAS 1 - PL/I

Not relevant.

PEARL

It is possible to put into a structured value as a component a pointer to another value of the same kind of structure:

TYPE LINK = STRUCT (CONTENTS STRUCT (anything).

POINTER REF LINK);



PL/I

By use of ALLOCATE and FREE statements, BASED, POINTER, and OFFSET variables and the NULL built in function.

PROCOL

Not possible to use pointers.

RTL/2

By assigning the name of one cell to a reference component of another cell.

For example using the example of 43:

CELLA.TL:=CELLB;

where

CELL CELLA,CELLB;

will link

CELLB to CELLA.

47- What kind of list processing features are present?

ALGOL 68

Storage generators (loc and heap) reference comparison using is, isnt and nil.

CORAL 66

None.

PAS 1 - PL/I

Not relevant.

PEARL

See answer to 46.

PL/I

As 46.

PROCOL

None in the T2000 version.

RTL/2

No explicit features but see answer to 43 and 46.

48- How are new data types defined?

ALGOL 68

User may define new modes in terms of those previously declared,  
e.g. mode element = union (int, ref cell)  
mode cell = struct (element head, tail).

CORAL 66

Not possible to define new data types in ALGOL 68 style using  
mode.

PAS 1 - PL/I

Not possible.

PEARL

It is possible because each different structured entity is a  
new data type if it is treated as a whole.

PL/I

By structure definitions or preprocessor procedures.

PROCOL

Not possible.

RTL/2

For example MODE COMPLEX (REAL RL, IM);

49- How are new key-words, which may be treated as data attributes in declarations, defined?

ALGOL 68

As above in 48.

CORAL 66

Not possible to define new key-words.

PAS 1 - PL/I

Not possible.

PEARL

It is possible using the TYPE declaration, e.g.:

TYPE COMPL =SRTUCT (RE REAL, IM REAL).

This introduces COMPL as a new key-word which may be used in subsequent declarations in the same manner as REAL or INT or BIN and so on.

PL/I

Preprocessor procedures.

PROCOL

Not possible.

RTL/2

COMPLEX in answer 48 is essentially a new key-word.

50- How are labels placed in aggregates (label arrays or structures)?

ALGOL 68

Labels cannot be manipulated as objects in ALGOL 68. However, they can appear in procedure bodies which themselves can be placed in structures or passed as parameters to other procedures.

CORAL 66

'SWITCH' is a one dimensional array of labels and is declared, and always initialized, as follows:

'SWITCH' JUMPLIST := L1, BL48, ERREXIT, EXIT 1, EXIT 2;

PAS 1 - PL/I

One dimensional arrays of labels are allowed. They must be initialized.

PEARL

It is possible to build up label arrays (either of label constants or of label variables).

PL/I

By assignment or by use of the INITIAL attribute.

PROCOL

Only a label-list (as in FORTRAN).

RTL/2

LABEL is a primitive mode and can be a component of a record or the type of an array. Such use is rare.

Example: MODE STRANGE (INT I, LABEL L);

ARRAY (1Ø) LABEL AL;

51- What are the default options when the context is too weak to define data?

ALGOL 68

There are no default options in ALGOL 68.

CORAL 66

No defaults.

PAS 1 - PL/I

Storage class AUTOMATIC, packed attribute ALIGNED, scope attributes.

PEARL

The problem is not treated in the language report (but see also 9 and 19).

PL/I

Defaults applied depend on the deducable contextual attributes.

PROCOL

Each data element must be declared explicitly.

RTL/2

No defaults.

Access to data

52- How is one and the same data item (storage) place declared with different identifiers?

ALGOL 68

Using identity declarations, e.g.: real x;  
  real y=x;  
x and y will then share the same storage.

CORAL 66

Using the 'OVERLAY' declaration.

PAS 1 - PL/I

Not possible.



PEARL

It is possible using the equivalent-attribute within the declaration. As an example the declarations

DCL SUM REAL: = Ø; DCL SALDO REAL = SUM;

effect that one and the same place (initialized with Ø) may be referred either by SUM or by SALDO.

PL/I

Using the DEFINED or BASED attributes.

PROCOL

Yes, by a LIKE statement.

RTL/2

Not strictly possible.

The LET, text replacement, facility allows preprocessing of names and in this way more than one identifier can refer to the same data item. Thus renaming of items is possible. See 2Ø.

53- Are there restrictions concerning the attributes of such identifiers?

ALGOL 68

Modes must be identical.

CORAL 66

Two dimensional array cannot normally be overlayed onto other items without special care. All other variable types can be overlayed but no correspondence checks is made.

PAS 1 - PL/I

Not relevant.

PEARL

The identifiers used within the equivalent attribute (in the example above: SUM) has to be declared with the same attribute (here: REAL) as the new identifier (SALDO).

PL/I

Yes, there are some detail restrictions particularly with regard to the INITIAL attribute.

PROCOL

There is restriction on the initialization of data when using the like statement but this imposes no restriction on the attributes possible.

RTL/2

See answer 52.

54- How is a value assigned to the place a pointer refers to using the pointer?

ALGOL 68

ref real a;

ref real (a): = 3.0;

The cast must be used on the left hand side to force dereferencing.

CORAL 66

Again, in CORAL one must utilize an 'INTEGER' variable as a pointer. An 'INTEGER' appearing on the left hand side of an assignment statement between brackets implies that it contains the address of the variable that is to receive the assigned value.

a:= 'LOCATION' (p);

[a] := 3;

is identical to p := 3;

PAS 1 - PL/I

Not possible.

PEARL

The identifier given to the pointer used as the left hand side of a normal assignment is sufficient to do the job. Automatic dereferencing will take place.

PL/I

DCL X BASED,  
Q POINTER:

.  
.  
.

or: DCL X BASED (Q);  
X=4

Q-> X = 4;

PROCOL

Not possible.

RTL/2

By preceding the reference identifier by VAL thus

REF INT A;  
VAL A:=3;

55- What are the default mechanisms?

ALGOL 68

The storage access mechanism is decided by context e.g. whether dereferencing is necessary or not. Explicit coercions can be achieved by the use of 'casts'.

CORAL 66

All access methods are implicit other than the explicit functions for obtaining the address of a variable and as shown in (54) using that address.

PAS 1 - PL/I

The access method is implicitly fixed. A variable stands for its address and is automatically dereferenced if necessary.

PEARL

The storage access mechanism is decided by context, e.g. whether dereferencing is necessary or not. Explicit.

PL/I

----

PROCOL

Access methods are implicit. For subscripted items, the mother P-structure address is computed first then followed by the item isolation mechanism.

RTL/2

A variable stands for its address and is automatically dereferenced if the context demands this.

56- How can the attributes of a data structure (e.g. dimensions of an array) be dynamically interrogated?  
(See 106 of present paper.)

ALGOL 68

Operators are provided for finding the bounds of a given dimension. The current mode of a union may be interrogated using a case conformity clause. For example:

```
mode rib = union (real, integer, boolean);  
rib x;  
case x in  
    (real)      :      c when x is real c,  
    (int)       :      c when x is int c,  
    (boolean)   :      c when x is boolean c  
esac
```

The current mode of x is set by assignment.

CORAL 66

Not possible to interrogate.

PAS 1 - PL/I

Checking of subscript ranges at run-time is optional.

PEARL

It is possible to interrogate dynamically the upper and the lower bounds of each dimension of an array using the operators UPB and LWB.

PL/I

Built-in-functions are provided to find the extent and the upper and lower bounds of a specified dimension of an array.

PROCOL

Not possible.

RTL/2

The length (upper bound) of an array is given by the operator LENGTH. Note that the lower bound is always 1.

57- How are the lower and upper limits of an index position at run-time defined?

ALGOL 68

In the declaration, e.g.: m:n, l:k+1 proc a;

If flex is in the mode by assignment, e.g.

mode string = flex 1:0 char;

The upper bound of string will change on assignment.

CORAL 66

As signed integer constants at compile time.



PAS 1 - PL/I

Not possible.

PEARL

It is possible as in ALGOL.

PL/I

By declaration of arrays and cross sections of arrays.

PROCOL

Not possible.

RTL/2

All arrays are static and the upper bound given as an integer constant in the declaration.

Example: ARRAY(12) INT AI;  
          ARRAY(2,3) INT A2;

58- What happens if an index value is outside the array bounds?

ALGOL 68

For arrays with non-flex bounds it is always an error. Whether or not the error is detected is implementation defined.

CORAL 66

Nothing, the conceptual item being accessed except for some implementations which allow subscript checking as an option. A hardware trap may occur if core protection is in use.

PAS 1 - PL/I

If a boundary check is demanded then when an error is detected, either a system or a programmed error reaction is performed.

PEARL

Not defined in the language proposal.

PL/I

Condition SUBSCRIPTRANGE is raised, if enabled.

PROCOL

This causes an error; detection is implementation dependent.

RTL/2

If array bound checks were not applied by the compiler then the action is undefined. If array bound checks were applied then an unrecoverable error will occur. Standard action is to call any monitor package and then pass control to the user's unrecoverable error label (ERL) with an error number (ERN) of 4.

Conditions under which checks are applied are defined in the language specification manual.

59- Can the program discover the bounds for an array?

ALGOL 68

Yes, using the operator's lwb and upb. When used as monadic operators they always refer to the first subscript positions. When used dyadically the left operand is the subscript position.

CORAL 66

Only if they have been inserted as macro names in the original source text.

PAS 1 - PL/I

No.

PEARL

Yes. Examples:

A: = 3 UPB ARR; / A becomes the upper bound of the  
3<sup>rd</sup> bound-pair of ARR /

B: = 1 LWB ARR;

PL/I

Yes.

PROCOL

No.

RTL/2

Yes. The lower bound is always 1. The upper bound is given by the operator LENGTH. This operation always applies to the 'first' subscript of a multidimensional array. To obtain other subscripts slicing must be applied. This is because all multidimensional arrays are treated as vectors of references to vectors.

60- What is the mechanism to select one single element out of an array?

ALGOL 68

Dope vector mechanism implied. Written as a  $[i,j,k]$  .

CORAL 66

Name the array followed by listing the subscripts which may be expressions.

Example: a  $[6, x+2y]$

obtains access to two dimensional array a.

PAS 1 - PL/I

Normal indexing.

PEARL

The mechanism is normal indexing.

PL/I

By subscript.

PROCOL

Normal indexing plus shift and isolation in the case of subscripted items.

RTL/2

The name of the array (or a REF array variable) is followed by a subscript list in brackets.

Example: AI(J)

A2(P,Q)

61- How is a row or a column of a matrix treated as a vector?

ALGOL 68

In the 'scope' of `[1:6, 1:6] real m;m [,3]` may be used to refer to a vector. The bounds of the sliced array may be changed using at.

CORAL 66

Not possible.

PAS 1 - PL/I

Not possible.

PEARL

Like ALGOL 68.

PL/I

As a cross section of an array.

PROCOL

Only possible when all of the elements of a row can be contained in one word - e.g. if M is declared as a 2 x 10 array each element of which is 1 byte long, there are 10 rows each consisting of one word.

RTL/2

Multidimensional arrays are compiled as arrays of references to arrays. A subarray defined by one of these references may be treated as an array of one less dimension in its own right. Hence considering the array A2 of example 57 as a matrix of 2 rows and 3 columns, the first row can be handled as an array of one dimension. Thus:

```
REF ARRAY INT RAI:=A2(1);
```

and RAI would then refer to the first row of A2. It is not possible to handle a single column.

Note A2(2) would point to row 2 of A2.

62- How are partitions of a matrix handled (elements, slices, blocks)?

ALGOL 68

```
m [3:4, 4:6]
```

would specify a partition of a matrix. Such a partition can be handled by assignment and by programmer declared operators.

CORAL 66

Not possible.

PAS 1 - PL/I

Not possible.

PEARL

Not possible.

PL/I

As cross sections of arrays.

PROCOL

Not possible.



RTL/2

See 61. No other partitions possible.

63- By what mechanism are the elements of a structure selected?

ALGOL 68

i of r in the scope of struct (int i, real x) r

CORAL 66

A 'TABLE' is a vector of fixed length records of the same format. Access to any element is made by quoting the name of the item required followed by the index to the record containing the desired item.

Example: using 'TABLE' declaration in question 40 and assuming I, CHARLES are some variables, reference is made to table as follows: I :=AGE [CHARLES] ;

PAS 1 - PL/I

Not relevant.

PEARL

In the example given in 40 the real number is selected by RS.VALUE and the character string by RS.NAME.

PL/I

By qualified name.

PROCOL

By the name of the item if no ambiguity; if ambiguity is possible the name of the P-structure precedes the name of the items:

example: STRU. ITE2 (I) ITE2 (I)  
STRO. ITE2 (I)<sup>or</sup>

RTL/2

The name of the record (or a REF record variable) is followed by a point (.) and the item name.

Example: U.RL

CELLA.HD

64- How are bits accessed?

ALGOL 68

Bits may be declared and accessed in two different ways in ALGOL 68:

- 1) As `[ ] bool`, in which case the elements of the array may be stored in adjacent physical bit positions or alternatively simply one per computer word (implementation dependent).
- 2) As `mode bits`, in which case the item declared will be a computer word treated as a bit string. The operator `elem` may be used to select individual bits.

CORAL 66

Either by declaring them in tables as part-words (see question 42) and referencing them as in question (63). Explicit bit reference can be made, e.g. `BITS [5,6] a := BITS [5,3] b + BITS [5,2] c`; Additionally, logical functions 'DIFFER', 'UNION', 'MASK' exist and can be used on any variable type and operate as XOR, IOR, AND in machine code.

PAS 1 - PL/I

Bit string data with a length of one bit may be declared. Single bit or groups of bits may be extracted by use of the substring function.

Example: `DCL B1 BIT(1), B16 BIT(16);`

`B1 = SUBSTR (B16, 5, 1);`

The 5<sup>th</sup> bit of B16 is assigned to B1.

PEARL

It is possible to define data only one bit long. Using the operators LBIT or RBIT one can select single bits out of data with mode BIN, e.g.

c: = aLBITb c becomes the a-th bit from the left out of b.

PL/I

Bit strings are a formal data type known to the language with specified conversion and operational rules.

PROCOL

A bit is a particular case of an "item" and is accessed directly by its name. It is necessarily embedded in a structure which is also separately accessible.

RTL/2

By using the logical operators LAND LOR NEV and logical shifts on integer values.

65- How are pointer contents accessed and modified?

ALGOL 68

By assignment.

CORAL 66

No explicit pointers. 'INTEGER' variables used as pointers are allocated new addresses using assignment and LOCATION function.  
Example: a:= 'LOCATION' (b [i] ); 'COMMENT' array item address;  
a:= a + 1; 'COMMENT' increments pointer;

PAS 1 - PL/I

Not relevant.

PEARL

To assign a reference as the value to a pointer, a normal assignment is used. The contents of a pointer is used (accessed) if the identifier of the pointer appears in a position, where a reference syntactically is expected. The syntax defines all possibilities explicitly.

PL/I

Modified by assignment. Can be used explicitly or implicitly. ADDR built in function will generate proper POINTER values. Note: OFFSET variables are treated similarly to POINTERS.

PROCOL

Not relevant.

RTL/2

By assignment.

Conversion of data

66- Which operators and built-in functions exist for the different data types and aggregates?

ALGOL 68

A very large number of operators and the usual mathematical functions are built-in to a fictitious outer block which surrounds the user program. Such a block is called the standard prelude.

CORAL 66

+ - x / 'DIFFER' 'UNION' 'MASK'  
'BITS' 'LOCATION' = 'AND' 'OR'  
< = > = < >

PAS 1 - PL/I

Operators: arithmetic, logical and relational.

Built-in functions: ABS, MAX, MIN, SIGN, SUBSTR, UNSPEC, COMPLETION.

PEARL

There is a four page list in the PEARL defining document showing all operators and operands.

PL/I

A very large number of operators and built-in functions are provided and can be used on almost all data types, any required conversions being done automatically.

PROCOL

"Arithmetic: +, -, x, /, xx,

Relational : EQ, NE, LE, LT, GT, GE,

Logical : NOT, AND, OR, EX, %  $\pm$  n .

RTL/2

Monadic:

+ - ABS NOT	null negate abs not
LENGTH	array length
BYTE INT FRAC REAL	mode conversion

Dyadic :

SLL SRL SHL	logical shift
SLA SRA SHA	arithmetic shift
* :/ // / MOD	multiply divide
LAND LOR NEV	logical
+ -	add subtract

Comparison :

= # < < = > > =	primitive
:=: :#:	reference



67- To what extent are mixed-mode expressions allowed?

ALGOL 68

To the extent to which the operators are declared. Indiscriminate mixing is not possible. Most usual combinations are allowed.

CORAL 66

Mixed-mode expressions not restricted.

PAS 1 - PL/I

Allowed with all arithmetic data.

PEARL

Mixed-mode expressions are allowed with all arithmetic data.

PL/I

Very few restrictions are made.

PROCOL

Allowed in certain cases....for example mixing of integer and shorteger type is only allowed in two operand expressions.

RTL/2

According to the types allowed as operands for the various operators. Generally allowed in practice.

68- List standard conversion procedures.

ALGOL 68

int string  
real string  
dec string  
dig char

ALGOL 68 (continued)

string int  
string real  
string dec  
char dig

CORAL 66

None.

PAS 1 - PL/I

Conversion is done by assignment or by conversion operator (see 70).

PEARL

A FIT B changes the precision of the first operand, A, to that defined by the second operand, B,

REAL	int - real
CHAR	int - string (1)
BINT	bin - int
BITS	int - bin .

PL/I

ABS, ADDR, BINARY, BIT, CHAR, COMPLEX, DECIMAL FIXED, FLOAT, OFFSET, POINTER, POLY, PRECISION, STRING, UNSPEC.

PROCOL

None.

RTL/2

BYTE INT FRAC REAL convert plain operand to given mode.

69- What types of procedures may be used as pseudo variables?

ALGOL 68

None.

CORAL 66

None.

PAS 1 - PL/I

The function procedures SUBSTR and UNSPEC.

PEARL

None.

PL/I

The built-in functions COMPLETION, COMPLEX, IMAG, ONCHAR, ONSOURCE, PRIORITY, REAL, STATUS, STRING, SUBSTR, UNSPEC.

PROCOL

None.

RTL/2

None.

70- Are there any automatic conversions by assignment? (If there are any, please list them.)

ALGOL 68

Assignment is a 'strong' position, therefore all forms of coercion can take place

- deproceduring
- dereferencing
- widening
- rowing
- uniting
- hiping

CORAL 66

The expression on the right hand side of an assignment statement

CORAL 66 (continued)

will be evaluated according to the type of the variable on the left hand side. Automatic conversion will occur where necessary. Facilities exist to allow the programmer to control automatic - conversions.

PAS 1 - PL/I

Yes, automatic conversion from binary fixed to binary float and vice versa, and from binary fixed to bit and vice versa.

PEARL

The problem is not treated in the language report.

PL/I

If the assignment is a legal one, then all necessary conversions are done automatically.

PROCOL

Yes, the type of variable on the left hand side may differ from the type of expression on the right hand side.

RTL/2

Dereferencing and widening.

71- How are aggregates, (e.g. Transferring arrays with one and only one assignment) assigned?

ALGOL 68

By use of the assignment operator (must have the same mode and bounds).

CORAL 66

Not possible.

PAS 1 - PL/I

Not possible.

PEARL

The identifiers of aggregates are used in the same manner as the identifiers of single or basic mode data are treated. Therefore, it is possible to assign the value of one array to another array with one and only one assignment.

PL/I

By a single assignment statement. NOTE: Multiple aggregates may be so assigned.

PROCOL

Not possible.

RTL/2

Not possible.

Transfer of data

72- What are the common conversion mechanisms for data to be transferred in an I/O operation (e.g. formatting)?

ALGOL 68

Format and formatless I/O is character based, the characters being translated to the external representation before I/O. In binary I/O no conversions are performed.

CORAL 66

None. I/O would normally be obtained by calling a procedure. Conversions would have to be programmed in the procedure. The parameter type would need to be passed along with its value. Non-standard parameter types are available which allows parameters to have the type and scale as well as value passed automatically to the procedure. This information would be utilized by the writer of an output procedure to select required conversion.



CORAL 66 (continued)

Another means of doing I/O would be by use of facilities in the operating system which could be accessed using macros which define the necessary entry mechanism. These macros are often automatically made available to a program.

PAS 1 - PL/I

Data are converted from internal to external form which is defined explicitly by format list or implicitly by the data-type itself.

PEARL

The transformation between the internal and the external representation of the information is mainly controlled by formats.

PL/I

Conversions can be done on all I/O either explicitly or implicitly between the external and internal representations.

PROCOL

Classical I/O as in FORTRAN.

Process I/O is described in the I/O paper.

RTL/2

There is no I/O in the language as such.

I/O is normally performed by calling standard procedures. The standard procedures normally available perform formatted and unformatted conversions to character streams.

73- How are such conversions described?

ALGOL 68

The conversion between internal and external representations of characters is made using the abs and repr built-in operators.

CORAL 66

See 72.

PAS 1 - PL/I

Conversion from internal to external data and vice versa is described by format strings.

PEARL

They are described by format-constants or format-variables.

PL/I

When required, by format lists, picture specification characters and the data types.

PROCOL

See 72.

RTL/2

See 72.

74- What can be transferred?

Names of data, contents, pointers, semaphores, etc..

ALGOL 68

Any mode which is made up of int, real, bool, char may be transferred. Names, semaphores, etc. may not be transferred. Structures may be transferred as long as the modes of the fields can be expressed in terms of the above modes.

CORAL 66

See 72.

PAS 1 - PL/I

The contents of arithmetic, bit and character data can be transferred.

PEARL

Only data contents may be transferred (if necessary after dereferencing).

PL/I

All data types.

PROCOL

None.

RTL/2

The standard set of procedures transfer BYTE, INT, FRAC and REAL values and REF ARRAY BYTE values.

Destruction of data

75- What is the range of validity of data names or data contents?

ALGOL 68

The 'usual' ALGOL rules with extensions for if...fi and others.

CORAL 66

Scope as for ALGOL 60. Any variable initialized is treated as an 'OWN' variable.

PAS 1 - PL/I

Names of EXTERNAL data are known everywhere. Other data are known in the block where they are declared and in all inner blocks where the name is not declared again.

PEARL

The block structure defines the range of validity of data names or data except where names or data have been declared with the attribute GLOBAL. Those are valid within the whole module.

PL/I

See 5.

PROCOL

The scope is the compiling unit.

RTL/2

Data in a data brick is static and permanently available. Local data follows normal block structure rules.

76- What are the mechanisms available to retrieve allocated resources (core, channels) after end of actual usage?  
(Redefinition of arrays, elimination of obsolete tasks, etc.)

ALGOL 68

There is no specific mechanism in language. Local storage is stack based, while heap storage must require a garbage collector.

CORAL 66

This is a system function and as such is not in the language. This functions would normally be accessed using macros as in question (72).

PAS 1 - PL/I

Allocated resources are freed automatically. Only standard peripherals may be reserved and freed explicitly by OPEN/CLOSE.

PEARL

Local storage is stack based. Routines or data may be declared as RESIDENT. Other resources are freed automatically due to the priority of the tasks.

PL/I

ALLOCATE and FREE statements for storage. OPEN and CLOSE for I/O files.



PROCOL

None.

RTL/2

No mechanism specified in the language.

Local data is stack organized and is automatically deallocated.

Simple statements

77- What are the precedence rules for operators?

ALGOL 68

Priorities are given numbers in range 1-10. All monadic operators have priority 10.

9  $\perp$   
8  $\text{lowb upb}$   $\uparrow$   
7  $\text{elem}/\div: \div*$   
6  $+ -$   
5  $> \gg \leq <$   
4  $\neq =$   
3  $\vee$   
2  $\wedge$   
1  $+=, ::, :=, /=, *=, -=.$

CORAL 66

As in ALGOL 60.

PAS 1 - PL/I

Same rules as in full PL/I.

PEARL

The operator precedence rules are exactly the same as in PL/I.

PL/I

Left to right within priority. Operators are classified into seven levels of priority.



PROCOL

Hierarchy: parenthesis: from inside to outside

inside a pair of parenthesis, for arithm. operations: \*\*, \*/; + -.

For logical operations: REL; NOT; EX; AND; OR and REL:=

GE, GT, LE, LT, EQ, NE.

RTL/2

Precedence in range 1-6. 6 SLL SRL SHL SLA SRA SHA

5 \* :/ // / MOD

4 LAND

3 LOR

2 NEV

1 + -

78- What kind of conditional expressions exist?

ALGOL 68

Any serial clause may deliver a bool result, e.g.:

(a:=2; b:=a; true)

and may be used as an element of an expression which will deliver a bool result. The operators and, or and not are defined for bool operands. Such an expression may be used within an if...fi statement to select either the then or the else alternatives.

CORAL 66

Example: 'IF' a > b 'OR' a = b 'AND' b <> 0 'THEN' x + 2y 'ELSE' x-2y  
with the boolean expression only being evaluated as far as necessary to determine the truth or falsity of the expression.

PAS 1 - PL/I

IF statement (see 86).

PEARL

The conditional expression is defined as:

IF binal (1)-expression THEN expression ELSE expression FI.

PL/I

See 86.

PROCOL

Conditional expressions are used in connection with the IF statement. They are restricted to the use of only one logical operator and REAL operands are not allowed.

RTL/2

IF condition THEN expression

[ELSEIF condition THEN expression] ...

ELSE expression END

where a condition is made up of comparisons connected by the words OR and AND of which AND is the more tightly binding.

Example: IF X=Y OR P<Q THEN A ELSE B END .

79- What kinds of assignments are possible (e.g. multiple assignment, assignment within expressions...)?

ALGOL 68

Both multiple assignment and assignment within expressions are possible.

CORAL 66

Single assignments.

PAS 1 - PL/I

Single and multiple assignment.

PEARL

Multiple assignment is possible.

PL/I

Single, multiple and pseudo-variable.

PROCOL

Only single assignment and shift statement. ( $A=B\% +M$ ; the variable A is assigned the value of B having been shifted right M places.)

RTL/2

Single and multiple assignments but not in expressions.

80- What restrictions are there on the use of labels?

ALGOL 68

Cannot jump into a block or a for loop. Labels cannot occur at the head of a serial clause until all the declarations are completed.

CORAL 66

Label is an identifier written in front of any executable statement. The scope of a label is the smallest block containing it. One cannot jump into a block.

Note: A label in an outermost block can be made 'COMMON' to other segments allowing special explicit breaking of above scoping.

PAS 1 - PL/I

The label must be an identifier. Labelling of declare-statements is not permitted. A statement may have several labels. No jumps into blocks or loops.

PEARL

The label must be an identifier. Only executable statements may be labelled. The scope is defined by block-structure and jumps into loops are impossible.

PL/I

Labels are names. "The appearance of a label prefixed on a statement constitutes explicit declaration of the label". Hence, restrictions on the use of labels are the same as for declarations (see 5, 7 and 8).

#### PROCOL

The label must label an executable statement and is written as an identifier; no declaration of labels is allowed. Jumps to a label within a loop or within an IF Statement are not allowed.

#### RTL/2

One cannot jump into a block. Labels must refer to executable statements and not to declarations.

81- How are label variables used?

#### ALGOL 68

Label variables can be simulated by the use of proc variables  
proc void next:= label;

#### CORAL 66

There are no label variables in CORAL but labels can be passed as procedure parameters. Labels are used to initialize 'SWITCH' vectors and also can follow the 'GOTO' key-word.

#### PAS 1 - PL/I

Identifiers may be declared as label variable to assign them labels as values.

Example: DCL L LABEL ;  
          L = M1;.....  
          GOTO L;.....  
          M1:.....

#### PEARL

It is possible to declare identifiers of type LABEL which may be handled similar to other variables. So it is possible to assign to a label variable or to compare two label variables (or labels resp.).



PL/I

Label variables can appear on both sides of an assignment statement, can be compared and can be used in a GOTO statement.

PROCOL

No label variables.

RTL/2

A label variable contains a label value which identifies an execution of a procedure and the address of a literal label in that procedure. A jump to a label value involves unwinding procedure calls as necessary followed by transferring control to the literal label. This process can fail if the value is out of scope and will cause an unrecoverable error.

Assignment and comparison of label values is possible.

82- Can one associate a branching point with a side effect of a program to accomplish an implicit change of program control?

ALGOL 68

No - except for I/O faults.

CORAL 66

No. But some implementations allow 'ON'-conditions.

PAS 1 - PL/I

ON-conditions for type conversion, overflow, zerodivide, endfile and transmit error.

PEARL

ON-conditions are possible.

PL/I

Yes, there are 22 standard ON-conditions plus the capability of user defined ON-conditions.



PROCOL

No.

RTL/2

Yes, typically for error recovery. For example stack overflow, illegal goto, arraybound failure, arithmetic overflow.

83- How are procedures, tasks or I/O operations given control and how do they give it back?

ALGOL 68

Procedures are "called" by using the procedure name optionally followed by parameters enclosed within brackets (all I/O operations are initiated by procedure call). A "task" is given control when the relevant element of a collateral-void-clause is executed. Procedures return control back to the point of call when the final statement of the procedure has been executed. Alternatively, the point of return may be changed by executing a go to. The same technique is used to return control from tasks.

CORAL 66

Untyped procedures are called in a statement consisting solely of the procedure call whereas a typed procedure call may occur in an expression.

A procedure call consists of the procedure name followed in brackets by a list of actual parameters, if any.

Example: FLOAT (FL, FX);  
a := SIGN (FX);

PAS 1 - PL/I

Procedures: CALL Proc. name (parameterlist);  
Functionprocedures: Appearance of the procedurename with parameterlist (optionally);  
Tasks: CALL of a procedure with event- and/or priority-option.

## PAS 1 - PL/I (continued)

Control is given back explicitly by END; or RETURN;

I/O-operation:        Appearance of PUT, GET, IN, OUT, READ, WRITE  
                         statement.

### PL/I

Function procedures: appearance of the procedure name with optionally, a parameterlist. A RETURN/END statement is used within the procedure.

Other synchronous procedures: A CALL statement without TASK, EVENT or PRIORITY options. A RETURN/END statement is used within the procedure.

Asynchronous procedures: A CALL statement with at least one of the TASK, EVENT or PRIORITY options. The associated EVENT variable is set complete on execution of the RETURN/END statement.

I/O: Appearance of GET, PUT, READ, WRITE, REWRITE, LOCATE, DELETE, OPEN, CLOSE, UNLOCK, DISPLAY statements with or without EVENT options. With EVENT option, the EVENT variable is set complete on completion of the I/O operation, otherwise the task is suspended until completion of the I/O operation.

### PROCOL

For procedures:        by CALL and RETURN statement.

For task:             by a task handling instruction (e.g. ACTIVATE, etc.).

### RTL/2

A procedure is called by following its identifier (or a procedure variable pointing to it) by a parameter list in brackets. Brackets must not be omitted if there are no parameters.

Example: OUT(X);

          X:=IN();

Tasking and I/O are performed via standard procedures.

Compound statements

84- Which are the language features available to define groups of statements to be executed sequentially?

ALGOL 68

begin....end may be used to delimit groups of statements. The statements should be separated by semicolons if the statements are to be executed sequentially.

CORAL 66

Blocks and compound statements both of which use 'BEGIN' and 'END' to group statements.

PAS 1 - PL/I

All statements are normally executed sequentially. Exception see 85.

PEARL

BEGIN....END or DO....DONE maybe used to group statements.

PL/I

DO....END delimit do-groups. BEGIN....END delimit begin-blocks. PROCEDURE....END delimit procedure blocks.

PROCOL

It is not a block structured language. 'DO....LOOP' define a block of sequential statements. Other outer blocks in which statements are executed sequentially are Procedure, Subroutine or Task.

RTL/2

Statements are normally executed sequentially.

85- Which are the language features available to define statements of independent execution (like parallel execution)?

#### ALGOL 68

Statements to be executed independently should be bracketed between begin...end and separated by commas.

#### CORAL 66

A CORAL program is the collection of statements which can be executed in parallel with another collection of CORAL statements. A program is a set of (one or more) independently compiled CORAL segments each of which is a block. Some compilers can produce re-entrant code although this facility may be a compile time option. A single copy of a procedure that has been compiled to re-entrant code may be used simultaneously by several programs.

#### PAS 1 - PL/I

Main procedures are performed independently. Procedures which are called with EVENT- and/or PRIORITY-option are performed as parallel tasks. Statement groups of PAS 1-programs which are enclosed in SIMDO, and END, are performed in parallel.

#### PEARL

Independent execution takes place only for independent tasks.

#### PL/I

Procedure blocks may be executed independently if invoked as asynchronous procedures; i.e. with TASK, EVENT or PRIORITY options.

#### PROCOL

Each unit of compilation can be executed separately.

#### RTL/2

Questions of parallel execution are strictly outside the definition of the language and depend upon the operating system.



RTL/2 (continued)

However, it is likely that the procedures describing the statements to be performed by separate tasks can be effectively executed in parallel. Note that all procedures are basically re-entrant and so a procedure could have many coexistent lives.

86- What are the conditional statements, what do they look like, and how can they be replaced by other language features?

ALGOL 68

if serial clause then serial clause else serial clause fi case serial clause in serial clause, serial clause...out serial clause esac.

One type of selection in the case statement can be the mode of a union. Some extensions are allowed, e.g. else if becomes elsif. Conditional statements cannot always be replaced by other language features.

CORAL 66

Conditional statement is of form:

'IF' condition 'THEN' any statement except a for statement or a conditional statement 'ELSE' statement;

Note that the statements following 'THEN', 'ELSE' may each be a block (see 84).

PAS 1 - PL/I

IF statement: IF comparison expression THEN statement block  
ELSE statement block.

ON statement in case of CONVERSION-error,

FIXED OVERFLOW

OVERFLOW (float variables)

ZERODIVIDE

ENDFILE

TRANSMIT-error.



#### PEARL

The conditional statement is defined:

IF binal (1)-expression THEN statement...ELSE statement...FI;  
its elaboration is the same as in other languages. The other conditional statement, the case statement: CASE integer expression IN a list of statements separated by commas OUT statement...ESAC.

The case statement serves to shorten nested conditional statements. In its simple form it serves as a computed goto, e.g.

CASE I IN L1, L2, L3, L4 ESAC;

that is a FORTRAN statement: GOTO (L1, L2, L3, L4). I.

#### PL/I

- The IF statement: IF...THEN...;ELSE...;
- The DO statement: DO I = 1 to 10, 11 WHILE (A=B);
- The SIGNAL and ON statements: ON condition-name...;  
SIGNAL condition-name;

#### PROCOL

- (a) IF condition THEN unconditional instruction;
- (b) IF condition THEN unconditional instruction ELSE unconditional instruction;  
condition : = A REL B logical expression with only one logical operator.  
unc. inst. is different from: DO, LOOP, SUBROUTINE, CODE, PROCOL, FORMAT, END, IF.

#### RTL/2

IF condition THEN sequence

[ELSEIF condition THEN sequence] ...

[ELSE sequence]

END

See 78 for condition.

Example: IF X=Y THEN

P:=Q; R:=0;

RTL/2 (continued)

```
ELSEIF X > Y THEN
    P:=Q:=0; R:=1;
ELSE
    Q:=P;
END;
```

Conditional statements cannot be replaced by other language features.

SWITCH expression OF labellist

Example: SWITCH I OF L1, L2, L3;

If the expression is out of range then control passes to the next statement otherwise to the relevant literal label.

A switch statement can be replaced by a many-way conditional statement.

87- What kinds of loops are programmable?

#### ALGOL 68

Full form of the repetitive statement is

for I from A by B to C while D do E

If I does not occur in D or E then for I from may be replaced by from;

from 1 by may be replaced by by, by 1 to by to, by 1 while by while and while true do by do;

Where A, B and C are unitary clauses yielding integral results, D is a unitary clause yielding a boolean result and E is a unitary void clause.

#### CORAL 66

'FOR' VARIABLE :=FORLIST 'DO' statement;

Note 1, the variable is exhaustively assigned each element of the arbitrary length FORLIST in turn.

Note 2, each element may be either

- 1) expression
- 2) expression 'WHILE' condition

CORAL 66 (continued)

3) expression 'STEP' expression 'UNTIL' expression  
where 2, 3 may cause several loops for the same FOR element.

Note 3, the statement may be compound (see 84).

PAS 1 - PL/I

DO-loop: DO variable = initial value BY increment TO final value,  
REPEAT-loop in PAS 1 only: a DO-loop with linear index-incrementation.

PEARL

There is a universal loop statement defined:

FOR identifier

FROM integer expression

BY integer expression

TO integer expression

WHILE binal (1) expression

DO statement...DONE;

The optional parts here are to be understood as follows:

From the complete form

FOR I FROM A BY B TO C WHILE D DO E DONE;

may the term FOR I be omitted, if I does not occur in D OR E;

If A is a constant with the value 1, FROM A may be omitted and assumed is FROM 1 by default. If B is a constant with the value 1, BY B may be omitted and assumed is BY 1 by default. If D is a constant with the value B'1' (true), WHILE D may be omitted and assumed is WHILE B '1'. If no upper bound is required, TO C may be omitted. This enables the implementer to produce optimal code for each kind of loop under the control of the programmer.

PL/I

The basic loop control facility is the do-group; i.e., a block of statements of the form DO;...END; The DO statement can be qualified by an iteration specification (e.g., I=1 to 10 by 2), a conditional specification (e.g., WHILE (A=B)) or a conditional

PL/I (continued)

iteration and conditional specification (e.g. I=1 to 10 by 2  
WHILE (A=B)).

PROCOL

DO-loops with another key-word for the end: LOOP.

RTL/2

There are two kinds of iterative statement rather like ALGOL W.

FOR I:=A BY B TO C DO sequence REP

is the normal cyclic form. If B is a constant of value 1 then

BY B may be omitted. If I does not occur in the sequence then

the form TO C DO sequence REP may be used.

WHILE condition DO sequence REP is the second kind. This repeats  
the sequence so long as the condition remains true.

88- How can a loop be replaced by using other language features?

ALGOL 68

The section of code:

```
begin int J:=A, int K=B, L=C;  
  M: if if K>0 then J≤L else  
      K<0 then J>L else true fi  
      then int I=J; if D then E; J:= J+K  
      goto M fi  
  fi  
end
```

is equivalent to:

```
for I from A by B to C while D do E od;
```

CORAL 66

A loop can be programmed using conditional and GOTO statements.

PAS 1 - PL/I

By use of IF-statement.



PEARL

The exact effects of a loop statement may be described by a piece of program having the same effect:

The block

BEGIN

DCL J INT:= A, K VAL INT = B, L VAL INT = C;

M: IF IF K Ø THEN J = L

ELSE IF K Ø THEN J = L

ELSE B '1'

FI

FI

THEN BEGIN DCL I VAL INT = J;

IF D THEN E; J:= J+K; GOTO M; FI;

END;

FI;

END;

where the identifiers J, K, L, M do not occur in D or E, I differs from J and K. The operators , =, , =, + are standard operators, A, B and C are integer expressions, D is a binal (1) expression and E is a statement (or a sequence of statements), may be replaced by

FOR I FROM A BY B TO C WHILE D DO E DONE;

PL/I

By use of a begin-block containing conditional execution clauses and replacement assignment statements for the loop control variables.

PROCOL

DO I = A, B, C;

I = A

GO TO L1

LØ : I = I + C

L1 : IF C LT O GO TO L2

IF I GT B GO TO L3

GO TO L4



PROCOL (continued)

```
L2 :      IF I LT B   GO TO L3
L4 :      -
        -
        -
        -
        GO TO LØ
L3 :
```

RTL/2

Using conditional and GOTO statements.

89- Are there blocks of statements for dynamic loading (like BEGIN blocks)?

ALGOL 68

There are no facilities in the language for dynamically loading statements.

CORAL 66

There are no facilities in the language for this but it is allowed by some operating systems to handle the basic unit of compilation in this manner.

PAS 1 - PL/I

Core space for AUTOMATIC variables is dynamically reserved when a procedure is called.

PEARL

The code needed for execution of a specific task may be dynamically loaded after the execution of an ACTIVATE statement.

PL/I

FETCH and RELEASE statements allow naming of the procedures which can be used in a "load on call" environment.

PROCOL

None.

RTL/2

None.

90- How are new operators defined?

ALGOL 68

User may define his own operators. They must have declared priorities 1-9, e.g. op abs = (real a) real:

if a < 0 then - a else a fi;

CORAL 66

Must use procedures to implement new types of operation such as array addition.

PAS 1 - PL/I

Not possible.

PEARL

There exists an operator declaration. By such a declaration either an existing operator token may be used to be connected to another routine so that this operator may be used within variables of a mode which until now would be rejected by the compiler (e.g. the adding operator and variables of mode array), or completely new tokens or indicants (like identifiers for variables) may be introduced and connected with some special routine.

The following example shows how an operator declaration works.

Within the range of the two declarations

TYPE COMPL = STRUCT ( ( RE, IM ) REAL );

OPTR + ( COMPL, COMPL ) COMPL =

( (A, B ) COMPL )

RETURN ( ( A.RE + B.RE ), ( A.IM + B.IM ) );

the application is possible

PEARL (continued)

```
DCL ( Y, Z) COMPL := ( ( 1. 2 ), ( 3, 4 ) );
```

```
Z := Z + Y;
```

where after the elaboration Z refers to (4,6) or  $4 + i6$  as result of the complex addition.

PL/I

The functional capability of a new operator may be achieved by use of a function procedure. Such a procedure returns an evaluated expression to the point of invocation. The syntactic appearance can be made more "operator like" by use of the preprocessor capability.

PROCOL

Not possible.

RTL/2

Not possible.

91- How are procedures declared?

ALGOL 68

An item of mode proc is declared and associated with a routine body, e.g.

```
proc x = (real a) real: (routine body) in short form
```

or: 

```
proc (real) real x = (real a) real: (routine body)
```

in full.

CORAL 66

'TYPE' 'PROCEDURE' NAME (Formal parameter list);

Procedure Body;

The 'TYPE' is optional and is 'INTEGER'

'FIXED' (n, m)

or 'FLOATING'

The Procedure body is a single statement which may, however, be a Block containing its own local declarations.

PAS 1 - PL/I

An identifier followed by: PROCEDURE .... ;  
    identifier : PROC (parameter list);

PEARL

Procedures are declared in a declaration quite similar to all other variables, pointers and constants are e.g.

```
DCL UP PROC = (A REAL, B VAL REAL)
BEGIN A:= A + B; END
```

PL/I

By use of PROCEDURE and END statements.

PROCOL

There are two kinds of procedure: External and Internal

- Internal procedure:

    x TASK

```
SUBROUTINE MAT MUL (M1, M2);
```

Data declarations

```
PROC;
Procedure body
END;
```

External procedure:

    x REXT

```
SUBROUTINE MAT MUL (M1, M2);
```

data declarations

PROC

```
Procedure body
END
```

RTL/2

[ENT] PROC identifier (parameter description) [result mode] ;  
    body

ENDPROC

ENT, if present, indicates that the procedure name is to be externally accessible.

The parameter description lists the formal parameter by name giving their modes. Brackets are always present even if there are no parameters. The result mode, if present, indicates a function and its mode.



RTL/2 (continued)

```
Example:  PROC SQRT (REAL X) REAL;  
          *  
          *  
          *  
          ENDPROC
```

92- How are function procedures declared?

ALGOL 68

```
proc (,,, )  $\mu$   
 $\mu$  is the mode returned.
```

CORAL 66

See 91. (The use of a type makes a procedure a function procedure.)

PAS 1 - PL/I

Identifier: PROC (parameter list) RETURNS (attribute of result);

PEARL

The difference between a normal procedure declaration and a function procedure declaration is the appearance of a result attribute in the second case within the declaration.

PL/I

By the same mechanism as other procedures. Use of the RETURN (expression) is only allowed in function procedures.

PROCOL

Not possible in the T2000 version.

RTL/2

See 91.



93- What are the restrictions concerning the types and/or the number of parameters?

ALGOL 68

No restriction on number.

No restriction on types.

CORAL 66

The number and type of parameters in a declaration is not restricted.

PAS 1 - PL/I

Some restrictions for special procedures.

PEARL

There are no restrictions given in the language definition.

PL/I

No restrictions.

PROCOL

Maximum 15 parameters; no restriction on the type except no passing of 'items' and 'simple object'.

RTL/2

No restriction on the number of parameters. A parameter may be of any mode except whole arrays and records. It could be a REF array or REF record.

94- How are the attributes of the formal parameters specified?

ALGOL 68

The attributes are specified in the 'parameter pack'. See 91.

CORAL 66

See 91.

The formal parameter list appears to be very similar to normal variable declarations. However, any procedure appearing in the formal parameter list must also have details of its formal parameters given.

PAS 1 - PL/I

Formal parameters must be specified by declare-statements.

PEARL

Formal parameters are specified explicitly by a formal parameter list giving the identifier and the attribute for each parameter.

PL/I

By DECLARE statements internal to the procedure.

RTL/2

Within the parameter description in the procedure declaration.  
See 91.

95- How is the result type of function procedures defined?

ALGOL 68

See 91 and 92.

CORAL 66

See 91.

PAS 1 - PL/I

Result type specified by attribute of the procedure.

PEARL

The result type of function procedures is specified as attribute of the procedure identifier.

PL/I

By use of the RETURNS attribute on the ENTRY statement or by reference in the RETURN statement and by default from the initial letter of the PROCEDURE name.

PROCOL

Not relevant in T2000 version.

RTL/2

Following the parameter description list in the procedure declaration. See 91.

96- How is more than one identifier for one and the same procedure used?

ALGOL 68

Essentially by declaring a procedure "variable"

```
proc (real) real f;
```

```
f := g;
```

Call of f (x) and g (x) will be the same.

CORAL 66

Not possible. The overlay facility in CORAL is a means of re-using storage for a different purpose and since it is not possible to re-use a procedure in a different way then re-naming was not permitted.

PAS 1 - PL/I

It is possible to have several entry-points to a procedure but only one name is allowed for each entry. (\*)

PEARL

It is possible to declare another procedure identifier and to connect it to an existing procedure using the equivalent attribute in the same way as explained by 50.

PL/I

By use of the ENTRY statement or by multiple names on the PROCEDURE statement.

RTL/2

Not strictly possible but a similar effect can be obtained with the LET facility. See 2Ø.

Example: LET LOG=SQRT;

97- How are arrays of procedures handled?

ALGOL 68

Example: [1:4] proc switch  
= (goto e1, goto e2, goto e3, goto e4);  
Called by switch [4] .

CORAL 66

Not possible.

PAS 1 - PL/I

Not possible.

PEARL

Not possible.

PL/I

Aggregates of ENTRY variables are allowed. The entry-expression in the CALL statement must evaluate to a unique entry-constant at the time of execution of the CALL statement.

PROCOL

Not possible.

RTL/2

An array of procedure variables can be declared and initialized in the usual way.



RTL/2 (continued)

Example: `ARRAY(4)PROC(REAL)REAL F:=(LOG, SIN, COS);`

The procedure pointed to by an element of the array can be called by following the element name by a parameter list in the usual way.

Example: `I:=3;`

`X:=F(I) (X);`

in the above example is equivalent to

`X:=SIN(X);`

98- Explain or give the definition of the parameter mechanism.  
Is it language or implementation defined?

#### ALGOL 68

The mechanism by which formal parameters are identified with actual parameters is just as if the procedure body starts with

`Mode formal1 = Actual1,`  
`Mode formal2 = Actual2,`  
`etc.`

The effect is:

- 1) simple modes behave as value parameters
- 2) ref mode parameters behave as reference parameters
- 3) proc parameters behave as name parameters (in ALGOL 68 sense).

#### CORAL 66

Parameter types in CORAL 66 allow any object, whether a place, procedure or a data item to be passed to a procedure as parameters. In particular, data items may be called either by value or location. In both cases, the value of the actual parameter or the address of the actual parameter is evaluated once on entry and the formal parameter is initialized with the value of the actual parameter. This stipulation of evaluation only once is to obtain efficient object code.

Actual parameters corresponding to formal parameters called by value are not affected by assignments to the formal parameter within the procedure body. Actual parameters corresponding to



CORAL 66 (continued)

formal parameters called by location must agree exactly in type since assignments within the procedure to the formal parameter will directly affect the actual parameter.

PAS 1 - PL/I

By use of parameters address references are made, not value references.

PEARL

The parameter mechanism is language defined. Because the exact definition given in the report uses special terms from the PEARL syntax an example may give a slight idea:

```
DCL UP PROC = (A REAL, B VAL REAL)
      BEGIN A:= A + B; END;
```

The execution of the call-statement

```
UP ( X, 2 );
```

is easy to describe. Its effect is to replace the particular call by the sequence

```
BEGIN
      DCL A REAL      =X;
      B VAL REAL  =2;
      BEGIN A := A + B; END;
END
```

and then to execute these statements as if they would be standing in the place of the call UP ( X, 2 ).

PL/I

Argument passing is by name except for arguments without a name.

PROCOL

Like FORTRAN 'CALL by name'.

RTL/2

All parameters are handled as ALGOL 60 call-by-value. Thus the actual parameter is evaluated and assigned to the formal parameter. The formal parameter behaves like a normal local variable of the procedure.

99- How can a procedure be defined as re-entrant exclusive etc.?

ALGOL 68

Pragmats are used to supply implementation - specific information to compilers, for example

pragmat exclusive pragmat .

CORAL 66

There is no specified method of declaring procedures to be re-entrant. However, the appearance of a procedure in an external communicator could be enough to warrant it being compiled as re-entrant code. Also see question 85.

PAS 1 - PL/I

All procedures with the exception of PAS 1 procedures, OPTIONS (MAIN) and some special procedures are re-entrant.

PEARL

There is an attribute REENTRANT to be used in a declaration of a procedure to force translation in the way that re-entrability is guaranteed.

PL/I

RECURSIVE may be specified in the PROCEDURE statement. REENTRANT is commonly available under the OPTIONS clause of the PROCEDURE statement but is not defined in the language.

PROCOL

Only external subroutines can be shared by several task. They are not re-entrant and may be used as a resource.

RTL/2

All procedures are basically re-entrant.

100- List standard procedures.

ALGOL 68

Very large number for all I/O operations, e.g.

get, put, read, write.

See Chapter 10 of Report.

CORAL 66

None, but may be defined locally using supplied macros, etc.

PAS 1 - PL/I

There are 38 standard procedures to serve and check process-peripherals within PL/I programs.

PEARL

No such list given in the language defining document.

PL/I

There are 79 "built-in-functions" in the language.

PROCOL

None.

RTL/2

There is not a formal need to provide any standard procedures in an implementation but the following are recommended for I/O and error recovery.

IN OUT	stream variables
IREAD RREAD FREAD TREAD	read numbers/text
IWRT RWRT FWRT TWRT	write unformatted
IWRTE RWRTE FWRTE	write formatted
NLS SPS	layout
RRGEL	goto error label.

101- What procedures are there with reserved names?

ALGOL 68

None.

No procedures have reserved names although use of such names within an inner block renders the standard procedure inaccessible.

CORAL 66

None.

PAS 1 - PL/I

None.

PEARL

None.

PL/I

None.

PROCOL

None.

RTL/2

None.

102- What kind of facilities are available for setting strings of source code (MACRO prototypes) to be expanded in standard form at compile time according to a model (MACRO MODEL)?

ALGOL 68

Not available.

CORAL 66

CORAL 66 specifies a macro facility which allows parameters, and nesting of macro definitions to any depth, and further specifies deletion and re-definition should be possible.

AD-A032 571

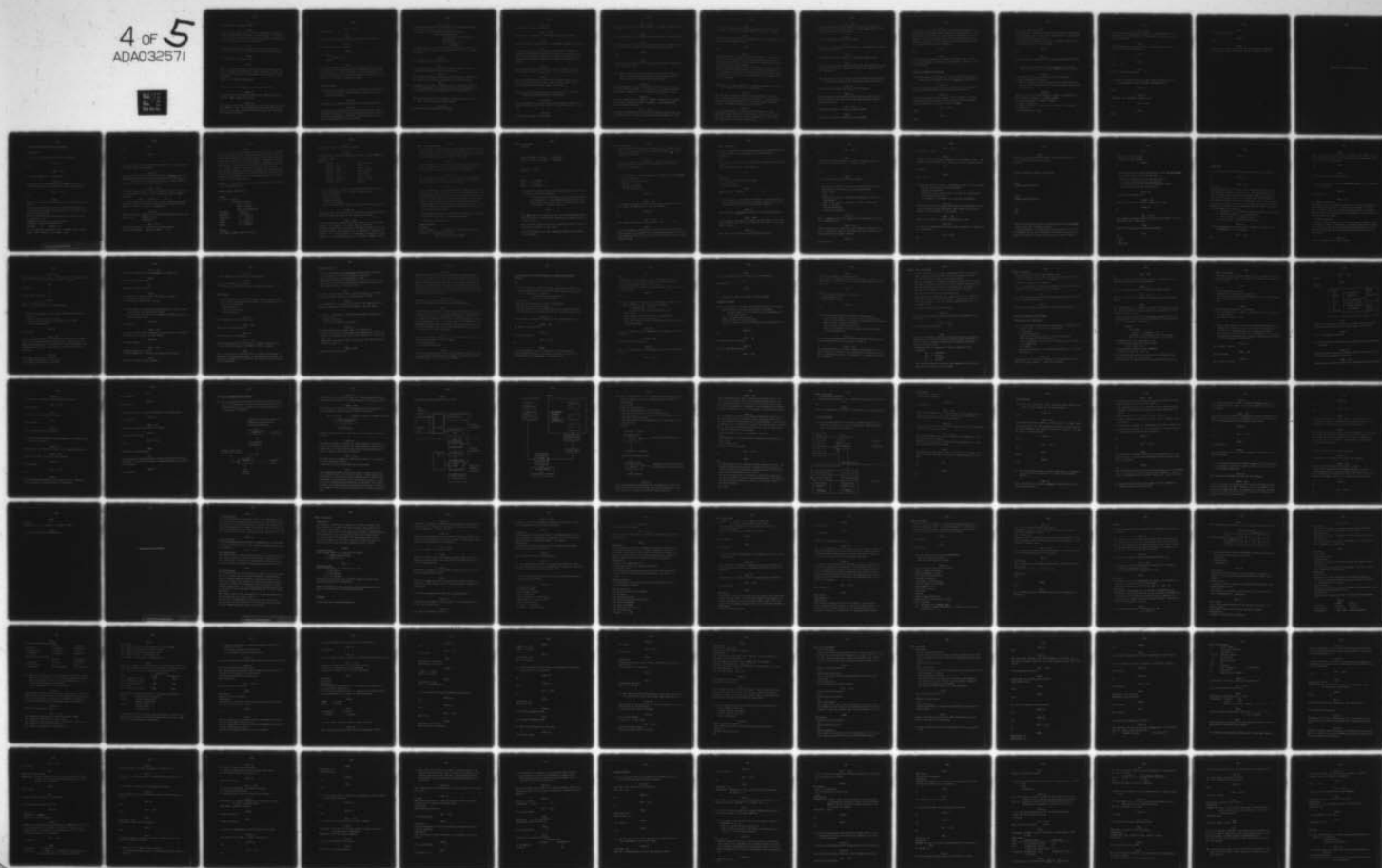
PURDUE UNIV LAFAYETTE IND PURDUE LAB FOR APPLIED IND--ETC F/G 9/2  
A LANGUAGE COMPARISON. A COMPARISON OF THE PROPERTIES OF THE PR--ETC(U)  
OCT 76 R ROESSLER, K SCHENK

N00014-76-C-0732

NL

UNCLASSIFIED

4 of 5  
ADA032571





PAS 1 - PL/I

No facilities on language level.

PEARL

There are none other than the TYPE and OPERATOR declarations which are not MACRO facilities in the sense that a precompilation is done to change them into legal language statements but could be regarded as such by their power.

PL/I

The preprocessor facility allows conditional replacement of source text strings and insertion of external text strings.

PROCOL

Only insertion of assembly code.

RTL/2

There is a simple facility hardly deserving the name of macro which is primarily intended for naming constants. It can, however, be used for general replacements but it does not allow parameters.

Example: LET RAB=REF ARRAY BYTE;

103- How is in-line assembly language code handled?

ALGOL 68

No provision in the language although most compilers allow it with the code...edoc construction.

CORAL 66

The reserved word 'CODE' introduces in-line code which is written as a 'BEGIN' - 'END' block immediately following. This code is often put in a macro definition with the macro name giving the code some significance.

PAS 1 - PL/I

Not possible.

PEARL

There is nothing defined in this direction in the report.

PL/I

Not specified by the language.

PROCOL

Yes: CODE;  
assembly code  
PROCOL;

RTL/2

In a code sequence starting with the key-word CODE and ending with the key-word RTL preceeded by an implementation dependent trip character. Any RTL/2 identifier can be referenced in the code by proceeding it by this trip character. A second trip character is also used for checking purposes.

#### Use of programs

104- How is the language processed for execution? Which controls do exist to modify this processing: compiler or interpreter directives?

ALGOL 68

The question is outside the scope of the language definition.

CORAL 66

The language was designed to be compiled into machine code for subsequent execution. No language specified controls, such as conditional compilation, exist but the macro facility can be used to considerable and varied effect.

PAS 1 - PL/I

Programs are processed by a compiler, which generates machine code. There exist some compiler directives:

to provide or prevent double precision arithmetic,  
check of subscript range  
multiplication fixed-overflow  
zerodivide  
second parameter of  
SUBSTRING

Variables which have different names in different (external) procedures may be declared to be identical. (This is a loader directive.)

PEARL

Not defined in the language report.

PL/I

Not specified in the language. Both direct compile and fully interpretive implementations have been made.

PROCOL

The language is compiled. The compilation check is linked with the check from the system generation part and the loaded.

RTL/2

Programs are converted to machine code by a compiler. There is an OPTION statement which allows control over the insertion of array bound checks etc. depending upon the implementation.

105- Which resources are allocated at compile-time or at run-time, to hold program or data blocks?

ALGOL 68

All resources are allocated at run-time.

CORAL 66

No run-time allocation of storage.

PAS 1 - PL/I

Core storage is reserved at load-time for STATIC variables.

PEARL

Dynamic loading of routines after an ACTIVATE statement is possible.

PL/I

The programmer can indicate storage class STATIC to suggest to a compiler that dynamic storage allocation is not required. However, a good optimizing compiler can readily recognize many pre-allocatable situations regardless of specific action by the programmer.

PROCOL

Programs and data blocks are held on disk at compile time and in core when the task is activated (i.e. in the running state).

RTL/2

Static data (data bricks), procedures and stack areas are allocated at compile/link/load time. Local variable space is then allocated within the specific stack at run time.

106- How may some data organization be known only at run-time?  
(variable size arrays variable structures).

ALGOL 68

The organization of all data structures is known at compile time, although their size and life-time can only be determined at run-time.

CORAL 66

No run-time allocation of storage.

PAS 1 - PL/I

The organization of all data structures is known at compile time. See 104.

PEARL

The organization of all data structures is known at compile time.

PL/I

A programmer may use the asterisk notation or the VARYING attribute to indicate unknown array bounds and string lengths.

PROCOL

None.

RTL/2

All arrays and structures are static and their size is therefore known at compile time.

107- What are the facilities available to build up a single program out of segments prepared by different people with a minimum amount of communication between them?

ALGOL 68

Each segment must contain sufficient information to link it to the environment in which it is to be run (i.e. global variables, procedures, operators, etc.). Discussed in the Report 2.3(c).

CORAL 66

A set of communicators 'LIBRARY' 'COMMON' 'ABSOLUTE' and 'EXTERNAL' exist to allow non-local information to be communicated between program segments.

PAS 1 - PL/I

Different procedures may be written by different people who need not communicate to agree on the use of variable names. See 104.



PEARL

Different MODULES may be compiled separately and then linked together. Communication between MODULES is by use of GLOBAL.

PL/I

Procedures need only agree on EXTERNAL names and associated data types, and/or parameter lists.

PROCOL

None.

RTL/2

Separate modules are linked at the brick level. A brick may be marked ENT to indicate that it can be accessed from outside. A module which refers to bricks outside of itself must contain EXT descriptions of the bricks.

Only brick names need be communicated between different program writers. Obviously the format of a data brick and parameters of a procedure brick must be communicated but the actual names of variables in a data brick or parameters of a procedure brick need not be known.

108- How is a program adapted to a changing hardware environment?  
Which parts of it are to be rewritten?

ALGOL 68

The only way in which an ALGOL 68 program need to be changed is in its use of the I/O procedures. However, if the programmer is careful, making maximum use of the environment enquiries, hardware independent programs can be written.

CORAL 66

It is seldom necessary to rewrite CORAL programs to adapt to new environments since the macro facility can be used to generate the variable areas. In a new environment only the macros, held separately would need re-definition.

PAS 1 - PL/I

Standard peripherals may be changed at run-time. Changing of process peripherals means changing the ENVIRONMENT-part of PAS 1 programs and compiling them again.

PEARL

Only the SYSTEM part must be rewritten.

PL/I

Generally, a recompilation plus changes to ENVIRONMENT and OPTIONS options should suffice.

PROCOL

The COMMON unit: the definition of the environment parts.

RTL/2

If the program is well structured then only those modules directly communicating with the changed environment will need rewriting and recompiling.

109- Upon which class of machines is the language more efficient?  
Which hardware features are required for best efficiency?

ALGOL 68

The question is outside the scope of the language.

CORAL 66

The language was designed to produce efficient object code on small to medium size computers of the type normally used for real-time control.

PAS 1 - PL/I

No special conditions required by the language.

PEARL

No special conditions required by the language.

PL/I

The question is outside the scope of the language. However, implementations have been made of subsets of the language on fairly small machines including at least one small industrial control computer. In any case, the actual size of the executable code module is dependent on the machine architecture and the degree of optimization done by the compiler.

PROCOL

None.

RTL/2

The language was designed to produce efficient code on small to medium machines. 2s complement arithmetic is needed for maximum efficiency.

Possible extension of language

110- Which explicit facilities do exist to add new data modes, operations or statements using an existing implementation?

ALGOL 68

New data modes and operators may be added by declaring mode and op in terms of previously declared modes and operators.

CORAL 66

Existing facilities in the language can be grouped to produce a particular effect using the macro facilities, but extension is not possible in the ALGOL 68 sense of mode.

PAS 1 - PL/I

None.

PEARL

None.

PL/I

The compile time facilities:

- Any identifier appearing in the source program can be changed.
- The programmer can indicate which sections of his program are to be conditionally compiled.
- Strings of text residing in an user or system library can be incorporated into the user program.

PROCOL

None.

RTL/2

New data modes may effectively be added using MODE definitions.

111- Which missing features should be added in the language specifications to put it to the minimum level of a LTPL?

ALGOL 68

- 1) A method of associating interrupts with semaphores.
- 2) Some configuration language.
- 3) Possible extension of the language for the synchronization of parallel activities so that activities may be given names, priorities etc.

CORAL 66

Essential not to overload language to make it unsuitable for small machines. Extend in a modular manner.

- 1) Record facilities in place of 'TABLE'
- 2) Pointer variables.
- 3) Tasking features (if these can be defined in a language).
- 4) Standard I/O.
- 5) Multidimensional arrays.

PAS 1 - PL/I

Tasking facilities should be improved, clock-and duration variables should be available in PL/I programs, too (but facilities blow up the OS!).

PEARL

The development of the state of the art should be taken into account (e.g. critical sections).

PL/I

(\*)

PROCOL

None. (\*)

RTL/2

None in the language itself. (\*)

112- What facilities do exist to make external addresses known to the language (like some CAMAC implementations)?

ALGOL 68

None.

CORAL 66

'ABSOLUTE' and 'EXTERNAL' communicators.

PAS 1 - PL/I

None.

PEARL

None.



PL/I

The external attribute.

PROCOL

None.

RTL/2

None in the language other than the normal EX-ENT mechanism.  
May be done via the linker but this is outside the language.

CONFIGURATION AND SYSTEM DESCRIPTION

Linking Application Software to Hardware

Input/Output

1- Are I/O operations provided in the language?

ALGOL 68

Yes

CAMAC - IML

Yes (see answers to I/O questions)

CORAL 66

Not directly, but are handled via procedure calls or macros which are generally well defined for each implementation.

PAS 1 - PL/I

Yes

PEARL

There are "communication-statements" and "file-handling-statements":

Communication-statements exist for formatted communication and unformatted communication and both for files and devices.

The main key-words are:

MOVE (for not-formatted communication not involving files, e.g. process - I/O)

TRANSFER (like MOVE, but involving files)

READ/WRITE (formatted character I/O)

SEE/DRAW ( -"- graphic I/O)

The file-handling statements allow to CREATE, OPEN, CLOSE, DELETE, LOCK (protect), and UNLOCK a file.

PL/I

Yes

PROCOL

Yes

2- How are I/O devices symbolically identified in the application program? Please give examples.

ALGOL 68

I/O devices can only be referred to by their channels. All such channels must be known in the standard prelude. For example:-

open (file, "identification string", printer channel);  
would open a file on a printer channel.

CAMAC - IML

A device declaration introduces the symbolic name for the device at a stated CAMAC address, or the devices at a set of CAMAC addresses.

LOCD SCALER, H, 1, 1, 12, 0

is a local device declaration ( i.e. valid in this program segment only) for the identifier SCALER, at the Hardware address branch 1, crate 1, station 12, subaddress 0.

CORAL 66

Either as channel numbers or as strings, both being used as parameters, e.g.

SETSINK ("LP");

SETSOURCE (1);

PAS 1- PL/I

Standard peripherals : key-words or symbolic names  
process peripherals : symbolic names (\*)

PEARL

Not only devices, but also interrupts and signals are identified (in the same way as other program-elements) by identifiers chosen by the programmer and declared on one of the three reference-levels (0: constant, 1: normal variable, 2: address of normal variable) with one of the attributes DEVICE, INTERRUPT, or SIGNAL respectively. (Also arrays are possible and all types mentioned are - single or array - permitted as parts of structures). The connection between a declared identifier and the respective part of the system hardware is described by the "system-division" together with special related declarations within the "problem-division" of the PEARL-program.

Example of a "system-division" and some device declarations within the "problem-division"

MODULE LIBRARY PUMPPROCESS;

SYSTEM;

CP 1048 <-> CH372;

CH372\*1\*1,16<-ADC3;

\*3\*1,8 <-DID5;

\*4\*1,8 ->DOD2;

\*5\*1,8<->AUSGABE:FS3;

MANOMETER: ->ADC3\*57;

INDIKATOR: ->DID5\*72\*5,1;

VENTSTATUS: -> \*72\*6,2;

TASTE: -> \*78\*1,2;

PUMPE: <-DOD2\*53\*3,2;

VENTIL: <- \*53\*5,2;

PROBLEM;

DCL (TASTE, PUMPE, VENTIL) VAL DVC;

.  
.  
.  
.



#### PL/I

By symbolic name; e.g. READ FILE (name)

#### PROCOL

I/O devices are explicitly declared (defined) in the COMMON with a symbolic name.

Examples

DEFINE	LP.....	(line printer)
DEFINE	DK.....	(disk)
DEFINE	ANIN.....	(analog input)
DEFINE	ANOUT.....	(analog output)
DEFINE	MT.....	(mag. tape)
DEFINE	DIGIN.....	(digital input)
DEFINE	DIGOUT.....	(digital output)
DEFINE	SPC.....	(spécial)

3- Is it possible to connect these symbolic identifiers with external designations, e.g. vendor's designations?

- Is it necessary?

- When is it done?

- By what mechanism?

Please give examples

- How is it possible to modify such a connection at run-time?

#### ALGOL 68

This would have to be done in the Job Control Language and therefore is outside the scope of the language definition.

#### CAMAC - IML

It is necessary to connect the device identifier with the CAMAC address. The connection is finally done at run-time.

The device declaration can specify the CAMAC address as a single constant address (see example in 2) or sequence of constant addresses (i.e. a given array) or an arithmetic progression of addresses (i.e. a calculated array) or an indirect (CAMAC) address.

CAMAC - IML (continued)

For a given or calculated array, a particular member device is identified by a subscript which can be changed at run-time. For a device specified with an indirect CAMAC address, the address is a variable which can be changed at run-time.

CORAL 66

As the example in 2 shows this can be done if the system has been so implemented. As there are so many implementations it is not possible to answer the subquestions individually.

PAS 1 - PL/I

Yes, for process peripherals this is done in the "EQUIPMENT"-part (=system description), names of files can be linked to standard peripherals at compile or run-time.

PEARL

- It is necessary to connect the symbolic identifiers with implementation defined designations. In the "system-division" the programmer describes the parts of the system the program will use at run-time. These parts to which he wants to refer explicitly (in the "problem-division") have to be named by him.

These names can be used within the "problem-division" for several purposes. It is possible to define device arrays (e.g. for process applications).

- The mentioned "system-division" has to be given at "programming time". It is part of the source-program.

- Incomplete sample program:

MODULE EXAMPLE;

SYSTEM;

CPU<->CHANNEL;                    implementation defined designation

CHANNEL \* 3 ->PRINTER: LP # 701;

      name choosen by the programmer

.

.

.

PEARL (continued)

```
PROBLEM;  
.  
.  
.  
DECLARE VARIABLE  DEVICE;      (Reference 1)  
DECLARE CONST  VAL  DEVICE  = PRINTER;  
.  
.  
.  
VARIABLE: = CONST;  
.  
.  
.  
WRITE ... TO PRINTER ...;  
WRITE ... TO CONST . . .;  
WRITE ... TO VARIABLE ...;
```

- The specifications given by the "system-division" cannot be changed at run-time (evidently).

Device switching can be done e.g. by using assignments to reference - 1 - identifiers declared with the respective attribute or by using a variable index within a reference to a device - array.

PL/I

Yes

- no, some names are "reserved", that is, the system knows what kind of device is required and automatically assigns this file to that device.
- any time from system build time to execution time of first input/output statement to that name.
- various mechanisms are used depending on when the connection is established.

PL/I (continued)

- by CLOSEing the file-name then OPENing a new file-name directed at the same device or by various other operating system dependent facilities.

PROCOL

Yes, it is possible and necessary to connect these symbolic identifiers with the vendor's designations (e.g. LP, DK,...). See 2 for completeness.

4- Is it possible to specify hardware information about I/O devices?

- Is it necessary?
- What is this information?
- When is it given?
- By what mechanism?

Please give examples.

ALGOL 68

No

CAMAC - IML

No hardware information about I/O devices is specified explicitly in IML: the CAMAC specification is assumed.

CORAL 66

No

PAS 1 - PL/I

Only interrupt channels, working mode of ADC. (\*)

PEARL

The full hardware information of any device is implicitly contained within the implementation defined designations of the system components used for constructing the "system-division" of the PEARL-program.

PEARL (continued)

For each implementation these designations and informations have to be presented to the programmers in a system manual.

Therefore:

- No
- All hardware information concerning the respective system component.
- See the example given in the answer 3.

PL/I

Yes

- sometimes
- e.g., identify different disk types
- at various times
- multiple mechanisms

PROCOL

Implicitly, yes (DEFINE) (\*)

5- Is it possible to use existing features in the language for I/O devices which need special hardware interfaces not provided by the vendor of the application computer?

ALGOL 68

Only if special channels are present in the standard prelude.

CAMAC - IML

The features of IML apply for CAMAC devices whether or not the CAMAC-computer interface is provided by the vendor of the application computer.

CORAL 66

Yes, as all I/O is done via such existing features.



PAS 1 - PL/I

No

PEARL

Possible by "MOVE-statement". Transfer of command pattern as data until the point where O.S. support ends.

PL/I

Yes

PROCOL

Yes, mainly by use of the DEFINE statement.

6- Can these non standard devices be connected with symbolic identifiers in the same way as standard devices?

If 'no' then:

- Why?
  - What is the specific information necessary to make this connection?
  - What is the specific mechanism of such a connection?
  - When is it made?
  - By what mechanism?
- Please give examples.

ALGOL 68

Yes, if channels exist then files can be associated with them using procedure open.

CAMAC - IML

Whether devices are standard for the computer manufacturer or not is irrelevant. They can be handled by IML if and only if they conform to the CAMAC standard.

CORAL 66

Yes, as in 2.

PAS 1 - PL/I

Not relevant (see 5)

PEARL

Insofar as their individual components are related to O.S. - supported terminals which can be identified in the system-division.

PL/I

Possible

PROCOL

Yes, as in 2

7- Does the notion of 'default peripherals' exist in the language?

- How are switching facilities provided?

Please give examples.

- Which facilities are provided to enable the configuration of alternate data paths?

For example to accomodate I/O controller breakdowns.

ALGOL 68

The concept of default files exists within the language, but the actual peripheral types associated with these files (standin, standout and standback) are not stated.

CAMAC - IML

There is no notion of 'default peripherals' in IML.

CORAL 66

No, but some implementations allow a NULL peripheral or channel  $\emptyset$  to be specified.

PAS 1- PL/I

No

PEARL

- There only exist "standard-terminal" -declarations which are valid on the respective block level.

Example:

```
.  
.   
.   
DECLARE (PRINTER, TELETYPE) VAL DEVICE;  
.   
.   
.   
BEGIN;  
STANDARD WRITE PRINTER;  
.   
.   
.   
BEGIN;  
STANDARD WRITE TELETYPE;  
.   
.   
.   
END;  
END;  
.   
.   
. 
```

- There is no automatic device switching specified by the language rules in case of a hardware failure.

If the programmer is in doubt about the reliability of a system component he can use interrupts (or signals) and device variables (or arrays) to test one and switch to another device.

PL/I

Yes

- external to the language
- external to the language

PROCOL

No

8- Does the notion of 'system peripherals' exist with the language?

For example operator TTY, system disk,...

- Which are necessary to run the application?
- Can they be shared with the application?
- If "yes" then: how can the user 'organize' them?  
For example partitions on the disk,...

ALGOL 68

No, but see the answer to 7.

CAMAC - IML

There is no notion of 'system peripherals' in IML.

CORAL 66

No.

PAS 1 - PL/I

Yes, standard peripherals may be called by key-words, e.g. TYP = typewriter, TAP = tape punch

PEARL

There don't exist special system peripherals.

PL/I

Yes

- none
- yes
- many ways

PROCOL

No

Happenings

9- Are hardware or system Happenings provided in the language?

ALGOL 68

No

CAMAC - IML

Yes, both.

Hardware happenings are called 'LAM' (look at me) at crate level in CAMAC, and 'demand' at branch level. The system hardware can take a demand as a stimulus to investigate LAMs, and produce a combined bit-pattern called 'GL' (Graded LAM). The GL number is the ordinal number of the most significant 1 bit in the GL pattern.

An autonomous channel (hardware or software) can cause the normal sequence of execution to be interrupted and the current I/O transfer terminated as a result of

- a: completion of the specified number of words in a transfer
- b: buffer full/empty, i.e. internal limit
- c: end of block, i.e. external limit
- d: error status (x-response) from device.

CORAL 66

In some implementations the use of ON commands is allowed, e.g.:  
ON INTERRUPT (7) DO procedure.

PAS 1 - PL/I

Yes



PEARL

There exist interrupts and signals in PEARL. They differ in the way of programming reactions to them and in the respective attributes within the declarations.

PL/I

Yes

PROCOL

Yes (interrupts and software events).

10- How are hardware or system Happenings symbolically identified in the language?

ALGOL 68

Not relevant.

CAMAC - IML

a: Hardware happenings

IML assumes that the hardware responds to a demand by getting GL, then causing an interrupt. A declaration connects a GL number with a program label and a hardware address. When the interrupt happens, control is transferred to the program label connected to the particular GL number.

b: System happenings

A number of IML statements can include an escape parameter which is a label. If they are executed in a system with autonomous channels, then when the action specified by the statement has been carried out the program is interrupted and control goes to the escape label.

CORAL 66

By the ON command method when allowed.

PAS 1 - PL/I

Hardware-happenings (interrupts) are connected to software-happenings (event) in the "EVENT"-part of PAS 1 -programs.

PEARL

See answer 2

PL/I

By name within key-word.

PROCOL

By EV (n) or EVD (n) or IT  
where n is the index of the considered event.

11- How are these symbolic identifiers connected with hardware devices?

- When is such a connection made?
  - How can this connection be modified at run-time?
- Please give examples.

ALGOL 68

Not relevant.

CAMAC - IML

A bit in the GL pattern is connected with a station in a CAMAC crate by a hardware patching element called a LAM Grader. The CAMAC module plugged in at this station is then necessarily connected with this GL bit. The connection cannot be modified at run-time.

CORAL 66

Via numbers identifying channels.

- At program generation or start time.
- No, cannot be modified at run-time.

PAS 1 - PL/I

Symbolic description in the EQUIPMENT- and EVENT-parts.

PEARL

Nearly all the same answer 3.

PL/I

Externally to the language.

PROCOL

Via pointers to symbolic label identifying the device.

- At system generation time.
- No, cannot be modified at run-time.

12- Are hardware or system Happenings regarded as signal spikes or as setting and resetting switches?  
In the latter case, how is it possible to define logical relations between them?

ALGOL 68

Not relevant.

CAMAC - IML

A LAM is a signal level. A demand is a signal level. The Graded LAM pattern is a 24 bit wide bit pattern.

CORAL 66

As signal spikes.

PAS 1 - PL/I

Hardware-happenings = spikes  
software-happenings = setting and resetting of switches (\*)

PEARL

They are regarded as signal spikes.

PL/I

Both, depending on hardware and application.

PROCOL

They are regarded as switches.

EVD means the logical OR between an event and a delay. (\*)

Environment

13- Without duplicating material already covered, describe the way in which the hardware environment of the application software can be described?

- the attributes
- the flux of signals
- the connections

ALGOL 68

The language has no such facilities.

CAMAC - IML

There are no further details.

CORAL 66

Not relevant to the language.

PAS 1 - PL/I

Hardware description and connection to symbolic names is done in the system description (EQUIPMENT) of PAS 1 programs.

PEARL

This is done by the "system-division" in which the programmer describes the system environment of his application program (the system environment may also contain some special system software, not only hardware).

PEARL (continued)

- By the system-division connections between system components are described, not only the components themselves.
- The flux of signals is described by arrows.
- The connections must be constructed in a manner that for all parts of the specified system exists exactly one path to the CPU. This holds also for signals etc..

PL/I

By a fill in the blanks process, by macro-assembly for system build, by command language or by application program.

PROCOL

In the COMMON, all the peripherals used in the application are specified by means of declare statements , SYS and INOUT.

- 14- How is it possible to describe the necessary resources for a PA to run?
- When is it done?
  - Can it be changed at run-time?

ALGOL 68

The resources which a PA will need for running are:

- Made available by the programmer using semaphores so that only one PA, if necessary, may use a given resource at a particular time, and
- Resources such as core store, files, etc. are allocated dynamically by the O.S..

CAMAC - IML

IML does not deal with PAs.



#### CORAL 66

Either by routes and facilities described at system generation time or at program start time. Some resources such as core may also be allocated at run-time. The time at which resources are made available and changed depends on the type of resource. For example, a system might allow core to be allocated as necessary at run-time but only up to a limit set at system generation time. A message path to another PA may only be set up at system generation or PA initiation time.

#### PAS 1 - PL/I

Resources are allocated dynamically by the O.S.  
They may be reserved for single PAs using semaphores.

#### PEARL

Core memory and processing time management has mainly to be done automatically by the O.S. but the programmer has the possibility to attach the attribute RESIDENT to a PA when it is declared. For competition between PAs priorities can be assigned to the PAs at the activation (start) or continuation ( of a suspended task). Management of other resources can be done by using bolts and the operations RESERVE, FREE, ENTER and LEAVE which will be executed at run-time, or by using semaphores.

#### PL/I

Description mechanism depends on resource typewith the system keeping automatic track of certain types.

- at various times
- yes, for some resources.

#### PROCOL

A PA will obviously need logical and/or physical resources which are obtainable through semaphore request or through the system. The description is not statically achieved.

Linking Application Problem Software to Application System  
Software

PAS

15- Is it possible to specify general organization of PAs of a specific application by attributes other than those included in the language? For example:

- i) resident and non-resident PAs
- ii) mother-daughter structure

If yes, then:

- What are the components of such a description?
- When and how are they specified?
- Give the effects of such a description on the application operating system, e.g. modification of the size of system arrays.

ALGOL 68

This question is outside the scope of the language definition.

CAMAC - IML

IML does not deal with PAs.

CORAL 66

Not in the language.

PAS 1 - PL/I

No.

PEARL

It may be possible, but it is dependent on implementation. Mother-daughter structure, residence, priority of tasks are to be specified by the programmer on language level.

PL/I

Yes

- assignment of procedures to tasks and tasks to interrupts
- can be specified at system build or subsequent time by fill-in-the-blanks process or by macros
- total concurrent assignments may not exceed limits set at system build time.

PROCOL

No

16- Is it possible to describe a PA with other elements than those included in the language? For example:

- i) segmentation of a PA
- ii) size of the activation-stack of a PA

If yes, then:

- What are the components of such a description?
- When and how are they specified?
- Give the effects of such a description on the application operating system.

ALGOL 68

It is not possible in any implementation of which the writer is aware.

CAMAC - IML

IML does not deal with PAs.

CORAL 66

Not in the language but often possible in particular implementations.

PAS 1 - PL/I

No

PEARL

It is not possible since it seems not to be necessary.

PL/I

Not required.

PROCOL

No

17 - Question 17 was not included in this document.

Language features

18- Is it possible to specify restrictions in the use of certain language features in application PAs? For example:

maximum number of PAs waiting for the same Happening at the same time.

- What are these restrictions?
- When and how are they specified?
- Give the effects of such a destription on the application O.S..

ALGOL 68

No

CAMAC - IML

IML does not deal with PAs.

CORAL 66

Yes, in some implementation.

PAS 1 - PL/I

No

PEARL

Generally dependent on implementation.

Only the use of "Bolts" allows the programmer to specify a maximum number of tasks which may access the same resources at the same time.

PL/I

Yes

- Various queue sizes may be controlled
- at system build time
- reserves table sizes.

PROCOL

No.

19- Are software happenings provided in the language?

- How are they symbolically identified in the language?
- How are these symbolic identifiers connected with resources?
- When is such a connection made?
- How are they regarded by the O.S. ?
- How is it possible to define logical relations between software Happenings?

ALGOL 68

Yes. Dijkstra semaphore may be declared and used, both for process synchronisation and for mutual exclusion.

CAMAC - IML

The happening when an autonomous channel reaches the end of a block transfer is defined in IML regardless of whether the channel is implemented entirely in hardware or in software and hardware combined.



CAMAC - IML (continued)

- The I/O statement can include a parameter which is a program label. When the action specified by the statement has been carried out, control goes to the program label.
- The I/O statement can include a channel parameter which selects the type of channel to be used for the transfer. The connection between happening and channel is therefore by occurrence in the same I/O statement. No other resources are involved.
- The connection is made at programming time.
- Depending on the implementation, either the O.S. is assumed to save the program counter before going to the escape label, or the O.S. is unable to do this and the implementation is only a subset of IML.
- No logical relations can be defined.

CORAL 66

Yes, in some implementations facilities such as semaphores are provided. (\*)

PAS 1 - PL/I

Yes, events and semaphores. (\*)

PEARL

There are "semaphores" and "bolts" which facilitate process synchronization and mutual exclusion. Moreover there are possibilities to simulate hardware happenings via software statements. (Signals and interrupts).

- Identification of happenings is done by symbolic names.

Examples:

```
DCL  A  SEMAPHORE;  
DCL  B  BOLT(5);  
DCL  C  INTERRUPT;  
DCL  D  SIGNAL;
```

- The connection between resources and semaphores or bolts must explicitly be made by the programmer.

PEARL (continued)

- The connection is made at programming time.  
(Although changes at run-time are possible by the use of e.g.:  
file - variables or device - variables).
- Outside the scope of the language.
- It is only possible to specify an operation on a list of semaphore - or bolt variables (logical "AND").

PL/I

Yes, within the language and the operating system, they are identical to hardware happenings.

PROCOL

Yes. Used for process synchronisation and mutual exclusion. (\*)

Generating Application Software

Application O.S. generation

20- Is it possible to describe the application in order to build a specific O.S. for a specific application?

If yes, then:

- By what mechanism?
- What are the main elements of such a description?
- When and how are they specified?
- Is it possible to optimize O.S. modules, e.g. by additional assembling?

If no, then:

Show how a general O.S. is optimized according to a specific application.

ALGOL 68

The question is outside the scope of the language definition.  
ALGOL 68 is not a system generation language.

CAMAC - IML

No.

The O.S. is not affected. The IML program is optimized to the specific application by the programmer.

CORAL 66

Not in the language, purely on operating system feature.

PAS 1 - PL/I

The O.S. is tailored from different modules to specific application.

PEARL

The "system-division" in PEARL is a means for tailoring a specific (and optimized) operating system for an application out of a modular "master - O.S.".

- In the "system-division" the programmer defines the required configuration by means of device descriptions and the description of the hardware connections.

Example:

```
CPU<->CHAN;
```

```
CHAN * 5 ->TERMINAL : TTY (1);
```

(TERMINAL is the symbolic name for Teletype 1,  
which is connected to channel 5 of the CPU).

- The elements of such a description are
  1. Implementation defined "device types"  
(CPU, CHAN, TTY).
  2. User defined symbolic names (TERMINAL)
  3. Connection symbols (\*5, ->)
- The configuration description is made at programming time.
- Outside the scope of the language.
- The output of the compilation of the system description may

PEARL (continued)

deliver a set of parameters which allow to include (resp. exclude) the needed (resp. not needed) O.S. modules at system-generation time.

PL/I

Yes

- fill-in-the-blanks process
- parameters to a prototype application
- on f-i-b forms automatically generated in an iterative process from the prototype application
- Yes.

PROCOL

Typical system information are provided

- a/ in the COMMON part in order to tailor the O.S. to the specific application
- b/ at system generation time.

- 21- What is the general structure of an application tailored O.S.? Please draw a diagram to give a feel for the assumed O.S. required for an implementation as a language environment. Try to give a classification of O.S. modules (e.g.: standard/obligatory/optional/extension modules), displaying the effects of the configuration description.

ALGOL 68

Not relevant (see 20).

CAMAC - IML

Not applicable.

CORAL 66

Not relevant (see 20).

PAS 1 - PL/I

See 20.

PEARL

Example:

modules for handling of time and date etc. (optional)	file-handling (optional)	device driver (optional)
	O.S. nucleus for handling of basic language features (obligatory)	device driver (optional)
	. . . . .	

PL/I

Apart for a small initial kernel, all other nucleus tailoring decisions are arbitrary and no rigid layout can be shown.

PROCOL

A set of independent modules.

22- How are software facilities for simulating missing hardware included?

ALGOL 68

Missing hardware cannot be simulated, however in-core files can be set up using procedure associate.

CAMAC - IML

There are no software facilities for simulating missing hardware.



CORAL 66

Generally by simulation program specially written.

PAS 1 - PL/I

Not provided.

PEARL

These must be explicitly programmed.

PL/I

Not relevant.

PROCOL

Yes, floating point is simulated.

23- How is the file-handling system adapted to the application need (if required) ?

ALGOL 68

The question is outside the scope of the language definition.

CAMAC - IML

IML is at a lower level than file-handling.

CORAL 66

Not relevant.

PAS 1 - PL/I

(\*)

PEARL

The file-handling system may be optional but not changeable, only by means of "system-division" informations.

PL/I

Not relevant.

PROCOL

Not relevant.

24- How can the O.S. library be adapted to the application?

ALGOL 68

Not relevant.

CAMAC - IML

IML is not concerned with the O.S. library.

CORAL 66

Not relevant (see 20).

PAS 1 - PL/I

See 20.

PEARL

Dependent on implementation

PL/I

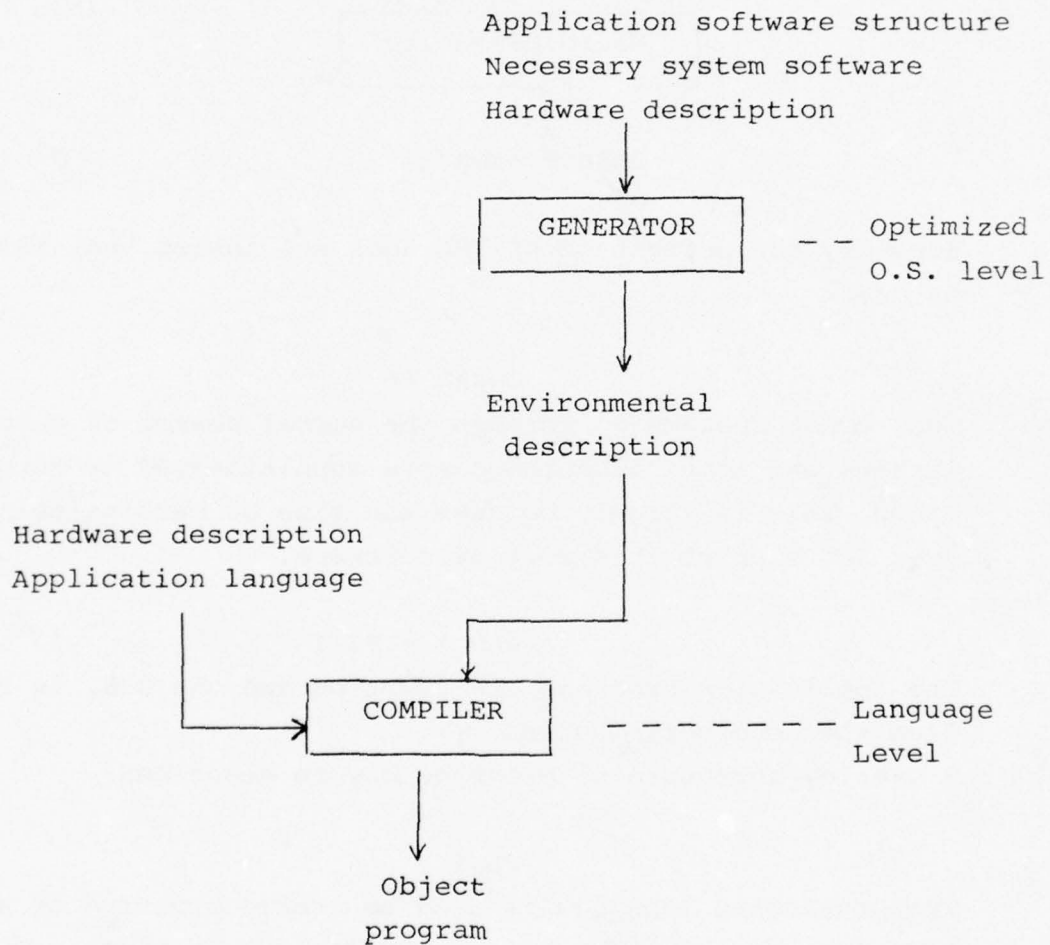
By an iterative cascaded process from the application prototype generation (a fill-in-the-blanks, assembly/compile and link-editing process).

PROCOL

Not possible.

Producing the Application Software

25- How is the application software produced and which parameters in the generation process can be influenced by the programmer? Please draw a diagram and indicate whether this is automatic. For example:

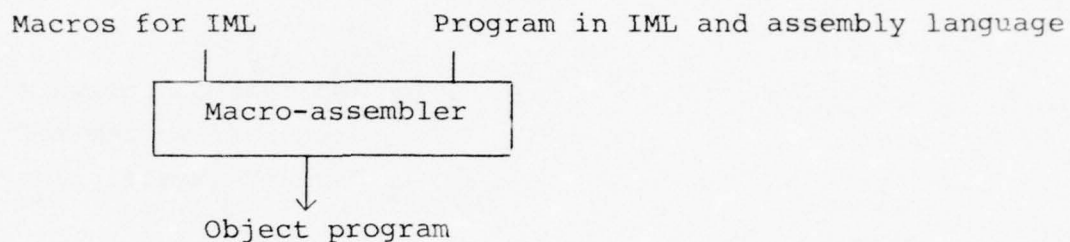


#### ALGOL 68

The question is outside the scope of a language definition and would be more relevant to a "customization" system such as APG/7.

#### CAMAC - IML

The application software is produced by manual programming. In a macro-assembler implementation of IML the steps are:



However, the definition of IML does not insist that this method be used.

#### CORAL 66

Most CORAL systems go through the normal phases of compiling, linking and loading. Nothing more sophisticated is normally provided. This is largely because the size of machine being used does not support this sort of software.

#### PAS 1 - PL/I

The application Programs are compiled and the O.S. is tailored from the necessary modules.

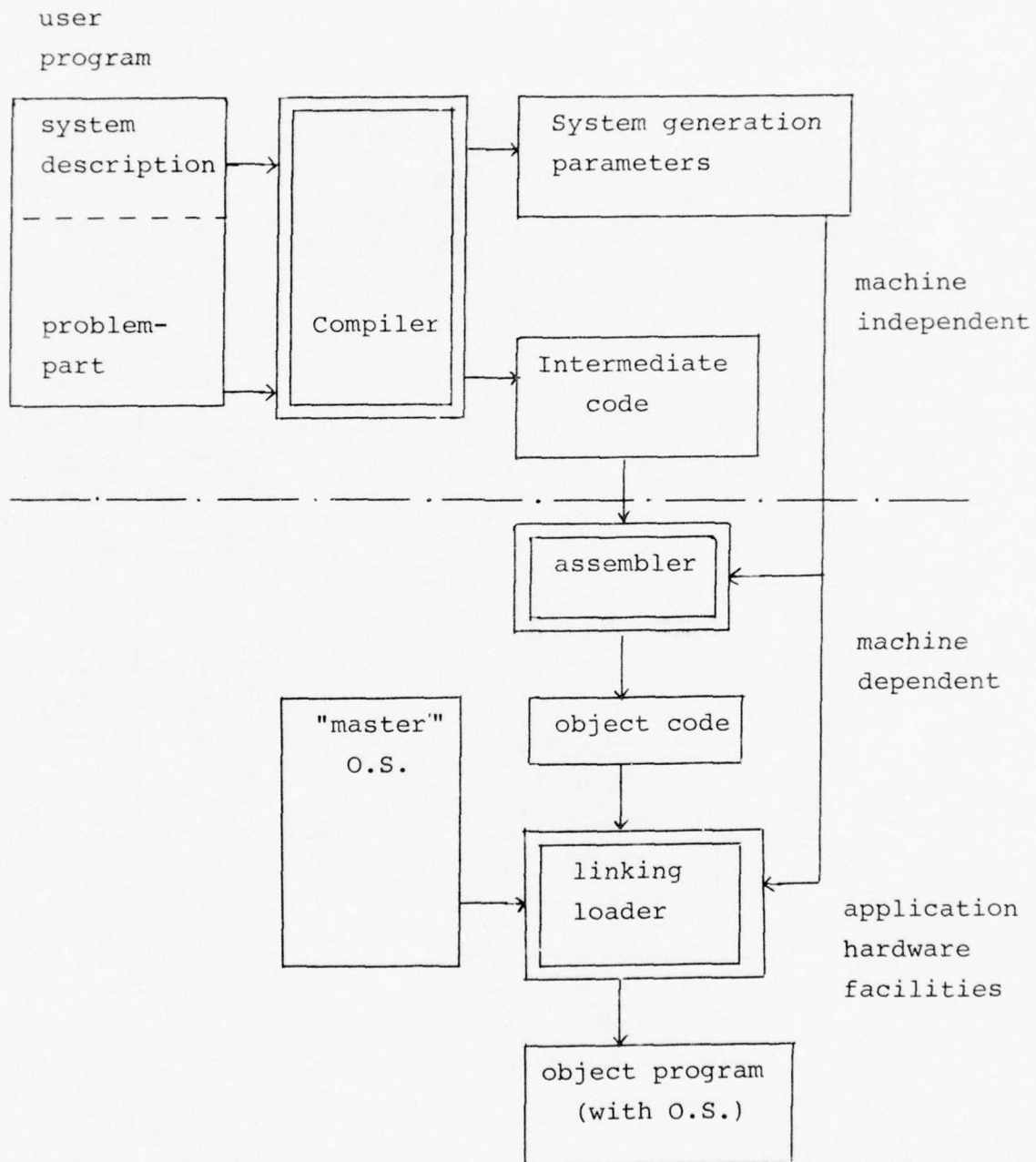
- Overlay structure of programs may be described.

#### PL/I

The programming language is used to create a series of application program modules. The characteristics of these modules are described by a fill-in-the-blanks language and an "application prototype" is constructed. As by the product of this, a series of fill-in-the-blanks questionnaires are produced. The answers to these are combined with the application prototype to generate the assemblies and link-edit procedures for a specific application.

PEARL

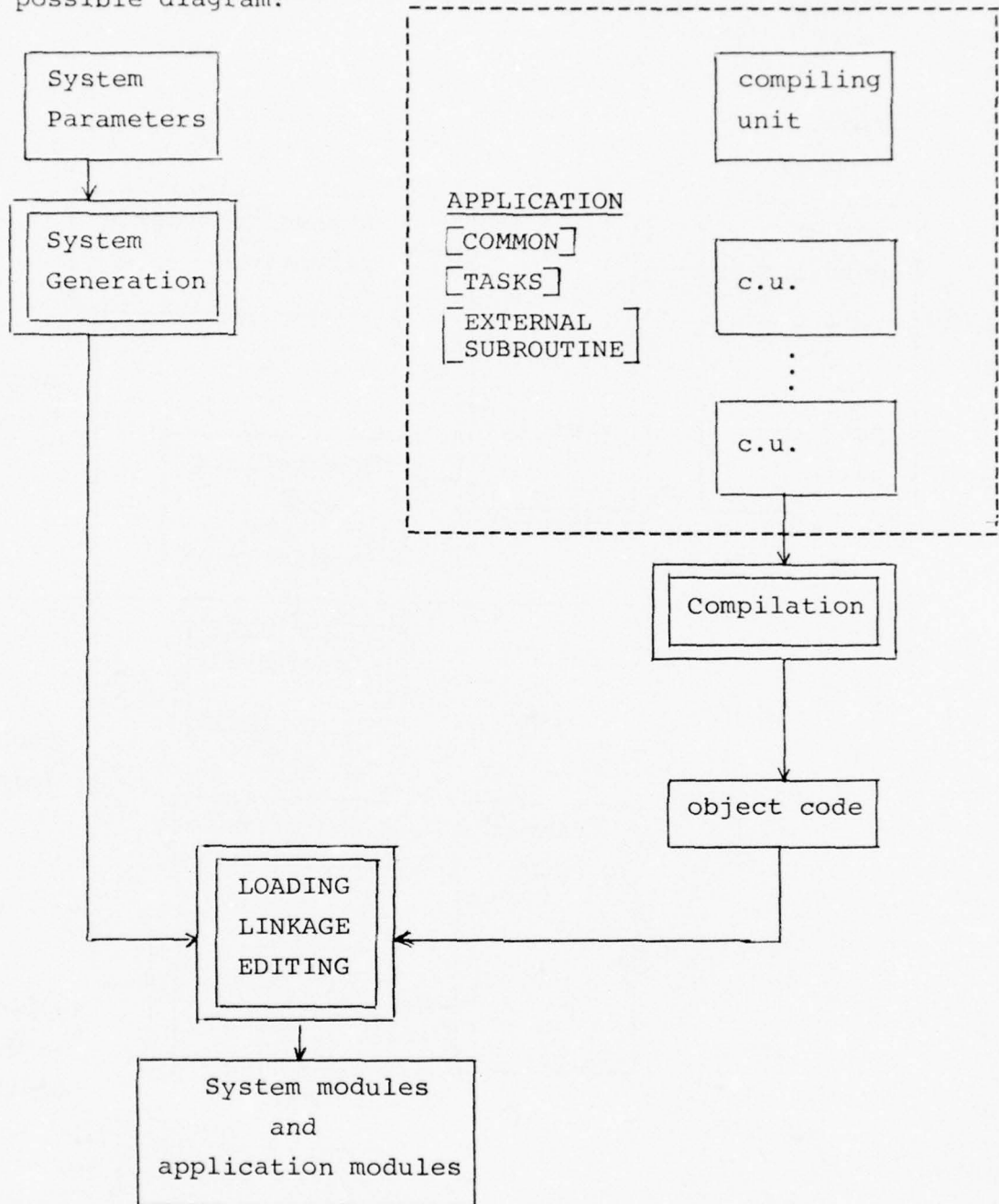
A possible structure may be the following:





PROCOL

A possible diagram:



26- Does a mechanism exist for a producing from the same source codes an object program for different object machines?

(See example).

If yes, then:

-What is the mechanism?

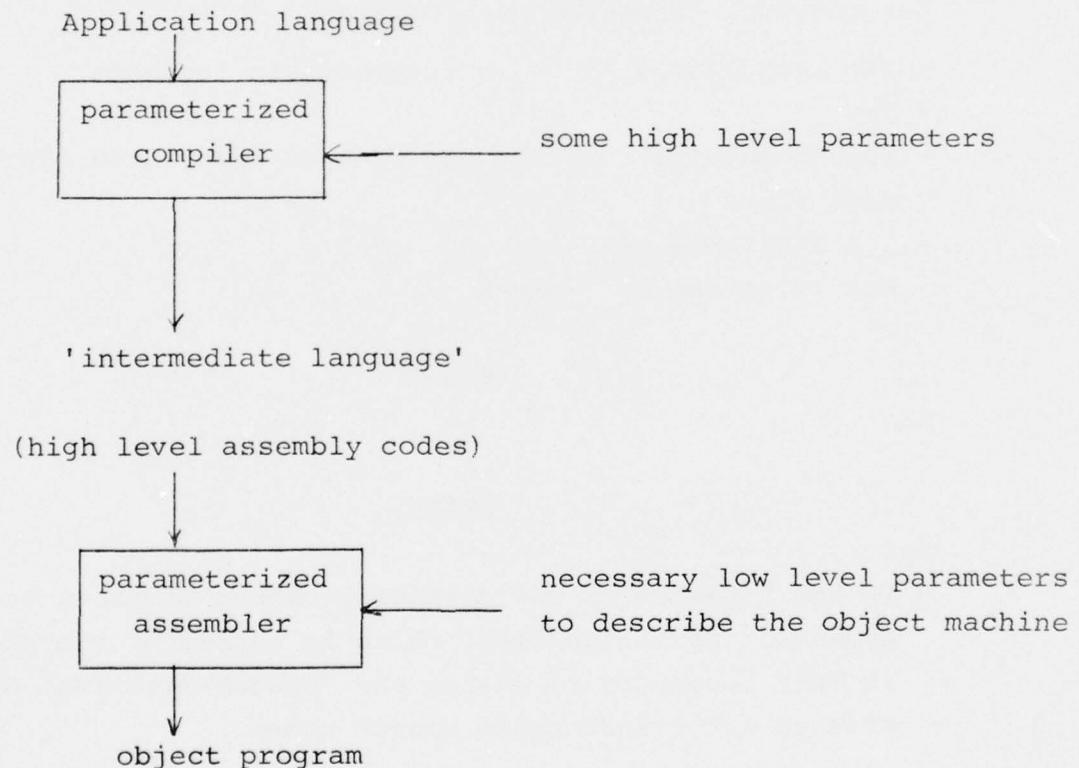
- Can the object machine be described?

- What are the elements of such a description?

- Can the generator, if exists, be 'parameterized' for that specific object machine?

- Can the compiler be 'parameterized' for that specific object machine?

Example:



#### ALGOL 68

Yes, using different code generators attached to the same compiler "front-end". The ALGOL 68C compiler has been used to produce code for a variety of different object machines.

CAMAC - IML

Each implementation of IML is expected to produce object code for a certain machine, different implementations produce code for different machines. The IML part of the source code would be machine independent; it depends on the implementation whether the data processing part of the program is machine independent.

CORAL 66

Not at present, although most CORAL 66 compilers are syntax driven from automatically generated syntax analysers which makes for some degree of transferability of the compilers. I am currently working on an intermediate language that can be used to produce object code for many machines and that can also be interpreted for powerful debugging of programmes.

- The mechanism will be an intermediate language
- Yes
- Pseudo-machine instructions with an associated dictionary for data items
- Not available yet
- Not if it can be avoided.

PAS 1 - PL/I

No.

PEARL

Yes

- If the hardware of the different object machines have the same capabilities in the field which is shared by the program, it is only necessary to change the "system-division" of the program to get transferable source codes.
- The object machine and its hardware is implicitly described by the device types within the system-description. All detailed information must be known at least by the linking loader of the object machine.
- See above

PEARL (continued)

- implementation dependent
- implementation dependent (for a possible example see question 25)

PL/I

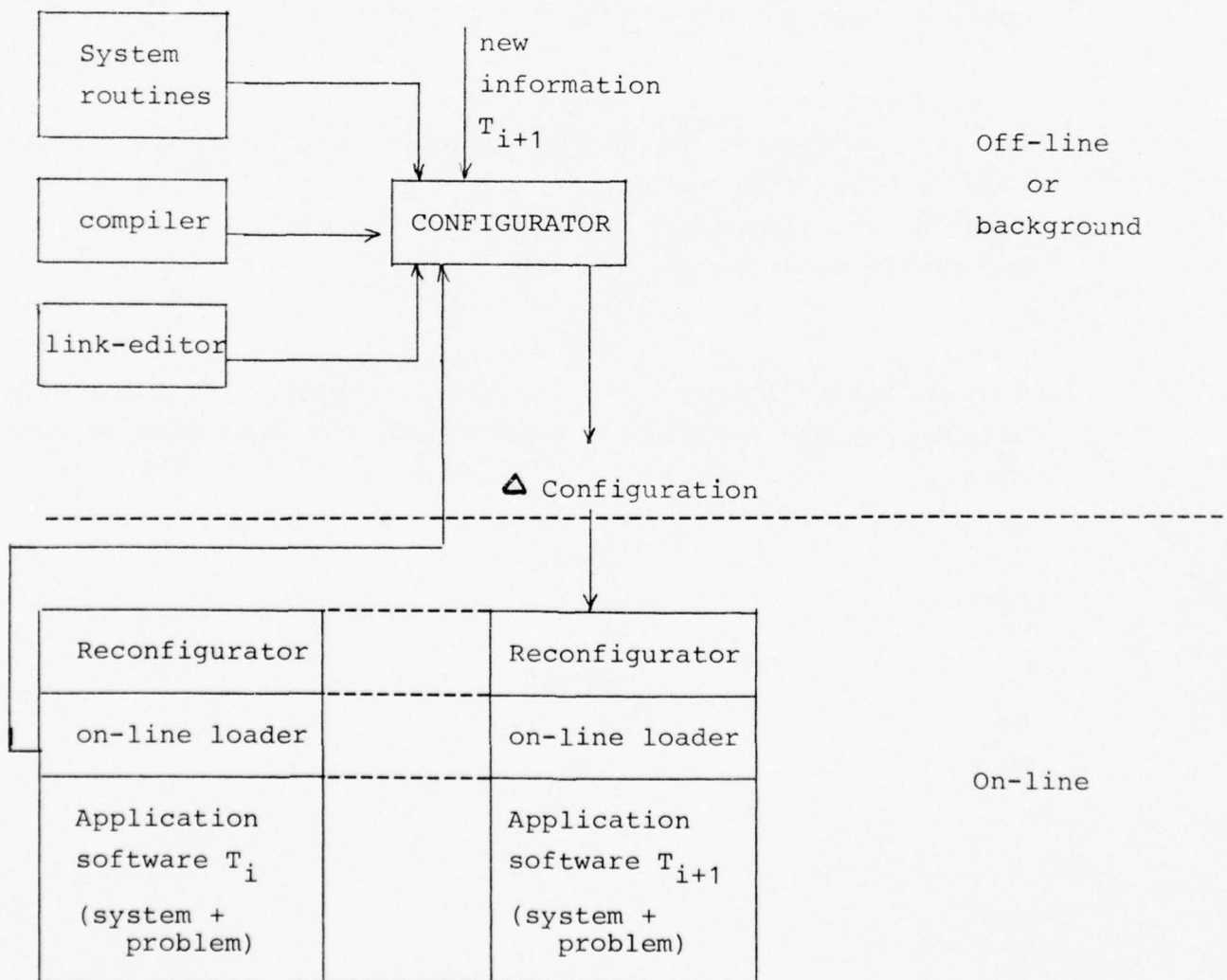
No.

PROCOL

Yes, by bootstrapping techniques. The compiler is not parameterized.

On-line updating

27- Does an application O.S. contain a standard function (e.g. like the following) for on-line updating an object-program?  
And if not, does it preclude it?



27-(continued)

What is this mechanism?

Please give a scheme.

ALGOL 68

Not relevant.

CAMAC - IML

IML does not assume that an application O.S. can update an object program on-line; while not specifically precluding it, it contains nothing that would make it easy.

CORAL 66

Not relevant as apart of a particular O.S. and one O.S. supporting CORAL 66 does not allow this. (\*)

PAS 1 - PL/I

Yes, the programmer may define variables as accessible at run-time by name or by number.

Moreover it is possible to access via the addresses of a load-map single words in the storage.

PEARL

Implementation independent. The module-structure of PEARL allows principally the addition or deletion of single modules at run-time.

PL/I

No.

PROCOL

No

No.



Miscellaneous

28- Please give information about important points which belong to but are not covered by this questionnaire.

ALGOL 68

None.

CAMAC - IML

IML defines the legal ranges for constituents of a CAMAC address. The specification of CAMAC addresses in device declarations may be partly symbolic (e.g. EXPTA for branch 1, crate 1), the symbol bound at translate-time.

CORAL 66

None.

PAS 1 - PL/I

None.

PEARL

None.

PROCOL

None.

29- Are all external devices or parts (modules) of a peripheral system explicitly addressable in terms of the language? Explain.

ALGOL 68

Only in as much as a different channel is defined for each of the eyternal devices.

CAMAC - IML

All CAMAC devices (modules) can be explicitly addressed in IML. A CAMAC address can be one of five types:

- a/ The combination B, C, N, A of branch crate, station and sub-address. This specifies a register or pseudo-register in a module.
- b/ The combination B, C, N which defines a module.
- c/ A crate address (B and C value only), used in special action statements.
- d/ A LAM source accessed at a subaddress (must have a BCNA value).
- e/ A LAM source accessed at a bit position in a status register ( must have BCN value and bit position I).

CORAL 66

No.

PAS 1 - PL/I

(\*)

PEARL

All locations of external devices may be described in the "system-division". Of course the implementation must provide respective O.S. support.

PROCOL

Yes

All referenced devices have to be defined explicitly in the COMMON unit: conventional as well as specialized peripherals are explicitly addressable one of several channels of a set of channels.

30- Explain how the routes (data paths) from the computer to external devices are identified.

ALGOL 68

All routes would have to have a different channel defined in the standard prelude. There is no way the user can define them for himself.

CAMAC - IML

The route to an I/O device is defined by a branch number B, crate number C, Station number N, subaddress A. This is its CAMAC address. Different devices have different BCNA values.

CORAL 66

(\*)

PAS 1 - PL/I

(\*)

PEARL

See question 2.

PROCOL

All routes would have to go through different channels for each external device.

31- Can a group of devices be handled together and if so is there a restriction on the data paths to them (e.g. only at adjacent hardware addresses?

ALGOL 68

This question is outside the scope of the language.

CAMAC - IML

Groups of devices can be specified for use in multiple actions, either with arbitrary CAMAC addresses or on an arithmetic progression of CAMAC addresses. Furthermore, all devices connected through one crate can be initialized, cleared or inhibited together, as can all on one branch.

CORAL 66

(\*)

PAS 1 - PL/I

(\*)

PEARL

One may use arrays of devices. There is no restriction on the data paths in general. Implementation dependent restrictions may be useful for the effectiveness of the drivers.

PROCOL

Since types of devices (Analog, Digital, Special) are distinguished within the I/O reference statement, it is not possible to address different types of devices in the same I/O instruction. For analog I/O, one of several channels may be referenced in the same I/O instruction.

32- How are sets of I/O devices declared?

ALGOL 68

This question is outside the scope of the language.

CAMAC - IML

A set of I/O devices can be declared as either a given array, enumerating the CAMAC addresses involved, or a calculated array, in which the CANAC addresses are an arithmetic progression with specification of the first, final and optionally increment (default 1) value for these addresses.

CORAL 66

(\*)

PAS 1 - PL/I

(\*)

PEARL

Example:

ANALOGINPUT (1:3):—→ADCA \* 3 + ADCB \* 2 + ADCC;

PROCOL

Each device is defined independently.



IMPLEMENTATION EXPERIENCE

## ALGOL 68

### Preliminary Note

All of the answers in this paper refer to an implementation of ALGOL 68 known as ALGOL 68C developed at the University of Cambridge by Dr. S.R. Bourne and his co-workers. The compiler has been bootstrapped onto a number of machines and it is therefore the implementation on one of these machines, namely the ICL 4130 at the University of York, that most of the information is given.

## CORAL 66

### Preliminary Note

In these answers only the modular 1 implementation is discussed. To produce answers for all of the current implementations would be impossible.

## PAS 1 - PL/I

### Preliminary Note

The answers refer to the implementation of the realtime programming language-system "PAS1-BBC-PL/I-Subset", consisting of a PL/I subset and on additional language PAS1 containing those features needed for realtime programming which are not in PL/I.

## PEARL

### Preliminary Note

In the moment the two most advanced (in terms of time) implementations of PEARL are those by BBC and ASME (= Arbeitsgemeinschaft Stuttgart-München-Erlangen). Both groups have not yet had time to adapt their answers to each other. Besides they deliberately planned to investigate different aspects of PEARL. Unified answers like e.g. for CORAL 66 will be provided as soon as experience from other PEARL implementations is be available.  
BBC-Subset:

The answers refer to the implementation of the BBC-PEARL-Subset (called PAS2 but not based on PAS1 !)

Two stages of implementation are planned. The first stage is already used for applications. in the second stage the subset will be completed by some additional features.

PEARL (continued)

ASME-Subset:

The compiler, described by answers as follows, represents the first version of a PEARL compilation system. The intention for developing this version is to give to the institutes concerned in a short time an aid to describe and test their "model processes". Slight changes in the language definition should be possible. In a second state of development all experiences made in the first state shall be collected to build an efficient PEARL compilation system (also from the commercial point of view).

PROCOL

Preliminary Note

These answers apply to 2 compilers for PROCOL

- i) T2000 (Telemecanique)
- ii) MITRA 15 (cii)

RTL/2

Preliminary Note

These answers apply to 3 compilers for RTL/2:

- i) DEC PDP-11
- ii) IBM 360/370
- iii) ICL System 4

Compilers (ii) and (iii) are virtually identical and will be treated together where appropriate.

Where the answers do not indicate to which implementation they apply, then they apply to all implementations.

Language

2- What was the language designed for?

ALGOL 68

"ALGOL 68 is designed to communicate algorithms, to execute them efficiently on a variety of computers, and to aid in teaching them to students" (Report Ø.1 (b)).

CORAL 66

CORAL 66 was designed for use in process control and real time systems. It is particularly suited to areas of application where object code efficiency is of importance.

PAS 1 - PL/I

For all purposes of process control.

PEARL

PEARL is a middle level programming language which allows to formulate the structure, the algorithms, time behaviour and I/O of real time programs.

PROCOL

The PROCOL language is a high level language for real time and process control applications.

RTL/2

RTL/2 is a simple high level language with features aimed specifically at applications such as data collection, communication and control.

3- Does the implementation meet any agreed standard?

ALGOL 68

The compiler implements a large subset of the language as defined in the Revised Report.

CORAL 66

Yes, with one permitted omission.

#### PAS 1 - PL/I

A subset of PL/I was implemented containing features for technical applications and realtime programming.

#### PEARL

##### BBC-Subset:

A medium subset of PEARL (defined in the "PEARL-proposal") has been implemented. Description of the subset is given in the language description "BBC-PEARL-SUBSET" (in German).

##### ASME-Subset:

A subset was implemented basing on the revision of the PEARL Report, published in October 1973.

#### PROCOL

Yes, the PROCOL standard specification.

#### RTL/2

Yes. All implementations meet the standard specified in 'RTL/2 language specification', RTL/2 document no.1, Version 2, published by Imperial Chemical Industries Ltd., 1974.

4- If the implementation does not meet any agreed standard which features were omitted?

#### ALGOL 68

- No parallel clauses
- No flexible names
- No formatted transput
- Use of indicants
- Round brackets in row-declarers
- Colon symbol in virtual rows
- Unit as actual row
- := and =: in operators
- widening of bits and bytes.



CORAL 66

Floating-point arithmetic.

PAS 1 - PL/I

Features of PL/I which are not needed for technical applications were omitted.

PEARL

BBC-Subset:

BOLT-variables, only binary SEMAphore-variables, LIBRARY-option of modules, dynamic creation of tasks, semaphores and files, structures, user defined data types and operators, graphic I/O, dynamic task priorities, recursive procedures, array operations, CASE-statement direct data transfer between peripherals.

ASME-Subset:

Structure of a PEARL Program:

- System division is not compiled separately
- No library names
- No recursion of procedure declarations and task declarations
- Not at compile time executable statements (only static initial)

System Division:

- No description of cross-linked process configuration (only tree-structure allowed)

Problem Division:

- No vectors used as data elements
- No pointers
- No structured values
- No type and operator declarations
- No vector assignment
- No label assignment
- No dereferencing operator
- Not excluded are:  
monadic +, -, NOT

PEARL (continued)

dyadic    +, -, \*\*, //, \*, /, AND, OR, EXOR, LT,  
          LE, EQ, GE, GT, NE, LBIT, RBIT, SHIFT

- No conditional expression (yet conditional statement)
- No case statement
- No bolt variables (yet semaphore variables).

PROCOL

Not relevant.

RTL/2

Not relevant.

- 5- Why was a subset implemented? (For example cost, core, size, usage, etc.).

ALGOL 68

Mainly because of manpower considerations and not because any omitted language feature was inherently difficult or costly to implement.

CORAL 66

Implemented on a computer without floating-point instructions.

PAS 1 - PL/I

See question 4.

PEARL

BBC-Subset:

Restrictions result mainly from the facilities of the operating system and the linker. In the second stage of development some of the restrictions will be removed (dynamic task priorities, structures and perhaps dynamic creation of tasks and semaphores).

ASME-Subset:

Saving of money and time.

PROCOL

Not relevant.

RTL/2

Not relevant.

6- Why omit particular features?

ALGOL 68

The main consideration was to be able to run a subset of ALGOL 68 that was suitable for writing compilers and for general systems programming work. Those features omitted were done so because they did not directly contribute towards this goal.

CORAL 66

In general, an implementation of CORAL 66 tends to reflect those features available in the hardware. Floating point was left out of the implementation because there was a floating point hardware. The official definition recognizes that certain features may be omitted from an implementation particularly when suitable hardware is not available and quotes the FLOATING point arithmetic as an example.

PAS 1 - PL/I

See question 4.

PEARL

BBC-Subset:

See question 5.

ASME-Subset:

The complete language definition for PEARL is too extensive to be implemented. The purpose of testing and demonstrating the applicability of PEARL especially for programming real time processes firstly it is necessary to implement these language

PEARL (continued)

features that are relevant to the above mentioned purpose, e.g. the task operation statements with mechanism for timing and scheduling, the interrupt operation statements, the synchronization statements and the process I/O statements.

PROCOL

Not relevant.

RTL/2

Not relevant.

7- Were any additional features implemented?

- Which features were added?
- Why were these features added?

ALGOL 68

Yes, the following features were added:

- Labels in enquiry-clauses
- upto and downto in loops
- until-part in loop clauses
- operator priorities
- row-symbol in row-declarer
- monadic formula as secondary
- Displacements (:= :=)
- Operate and assign
- Predicates
- Handles
- Radix of bits denotation
- Thef symbol in conditional- clauses

For two reasons:

- (i) Efficiency e.g. downto, upto
- (ii) Providing suitable language for a compiler writing environment e.g. := :=

CORAL 66

Yes, the following features were added:

REENTRANT procedures; EQUIV operator; binary and hexadecimal constants and preset strings.

The reasons are:

Reentrancy easy to implement on the particular computer; represents a machine code instruction; add to the flexibility of the standard language, resp..

PAS 1 - PL/I

Yes, semaphore variables in the PL/I-subset for synchronization of tasks and administration of resources.

PEARL

BBC-Subset:

Yes, variables of type EVENT and operations on these variables.

The reason:

An additional means for synchronization.

ASME-Subset:

None.

PROCOL

None.

RTL/2

No, of course not. All implementations meet the standard described in 3.



Compiler

8- On which machine does the compiler run and what is the target machine?

ALGOL 68

- The compiler runs on IBM 370/165 and ICL 4130. Code generators also exist for the PDP 11/45, CTL Modular I and other machines.
- The target machines are the same machines in each case, although as the code generators are written in ALGOL 68 any machine can be used as a code generator for any other.

CORAL 66

It is the same machine: Computer Technology Modular One.

PAS 1 - PL/I

- The compiler runs on IBM-computers on which a PL/I compiler is available, Honeywell 316, 516.
- The target machines are: Honeywell 316, 516.

PEARL

BBC-Subset:

- The compiler runs on IBM-computers on which a PL/I-compiler is available (not yet on PDP/11 but planned).
- The target machines are: PDP 11/05 /20 /35 /40 /45.

ASME-Subset:

- The machine independent part is running on SIEMENS 306 and AEG 60/50. Code generation only on the test machine SIEMENS 404/3.
- In principle all machines for which a Fortran Compiler and a code generator (STAGE 2) exists. Minimum hardware requirements have to be satisfied. (see 3.2)

PROCOL

It are the same machines: Telemecanique T 2000

C.I.I. MITRA 15

RTL/2

The combinations indicated by a X in the following matrix exist:

		Compiling machine		
		S 4	S 360/370	PDP 11
Target machine	S 4	X		
	S 360/370	X	X	
	PDP 11	X	X	X

9- State the minimum, and the recommended, hardware configuration required for compilation in terms of:

- Core store
- Backing store
- Peripherals.

ALGOL 68

- Core store:

On the ICL 4130 at least 24k (24 bit words) is required to compile a trivial program (excluding operating system components).

- Backing store:

A sequential file is needed for holding the intermediate code between passes 3 and 4.

- Peripherals:

Card reader (or sequential input medium) and some output device for messages e.g. lineprinter.

CORAL 66

- Core store:

16k or more words depending on the Operating System in use.

- Backing store:

None (paper tape); disk with two character streams.

- Peripherals:

Teletype; teletype and lineprinter.

PAS 1 - PL/I

-Core store:

150k bytes on IBM for all sizes of application programs.

16k words (16 bit) on Honeywell for very small programs.

- Backing store:

Sequential files for intermediate data and compiler passes.

- Peripherals:

One input device (cardreader, tapereader) and one output device (typewriter, lineprinter).

PEARL

BBC-Subset:

- Core store:

192k bytes on IBM.

- Backing store:

Sequential files for intermediate data and compiler passes.

- Peripherals:

One input device (cardreader, tapereader) and one output device (typewriter, lineprinter).

ASME-Subset:

- Core store:

Depends on the code optimizing of the Fortran Compiler.

For machines mentioned above the core store must be between 8 - 10k code and 8k for lists.

- Backing store:

Sequential and direct files are needed for intermediate results and holding some resident lists.

- Peripherals:

Cardreader and lineprinter and if used process peripherals.

PROCOL

	T 20000	MITRA 15
- Core store	12k core	14k core
- Backing store	1 disk unit	1 disk unit
- Peripherals	1 input and 1 output peripheral	

RTL/2

Compilers running on S4/360/370:

	Minimum	Recommended
-Core store	128k bytes	256k bytes
- Backing store	1 disk/drum	2 disks
- peripherals	something	something

Compilers running on PDP 11:

	Minimum	Recommended
- Core store	16k words	24k words
- Backing store	none	none
- Peripherals	PT reader/punch	PT reader/punch.

- 10- State any limits on size and complexity within the maximum program (e.g. number and size of arrays; number of constants; depth of nesting of blocks, loops and expressions) for:
- the minimum hardware configuration,
  - the recommended hardware configuration.

ALGOL 68

The compiler capacity is limited only by the size of store available to hold a "tree" version of the program as it is transformed during compilation. A tree size of 10k will accommodate a program of about 700 lines. Therefore there are no limits.

CORAL 66

There are no practical limits.

PAS 1 - PL/I

Max. number of constants and identifiers: about 2000,  
max. nesting of blocks and loops: 10 each,  
max. number of characters per name: 10 but only 6 significant,  
max. number of bits in bitstrings: 32,  
max. number of characters in strings: 127.



PEARL

BBC-Subset:

Max. number of constants and identifiers: about 2500,  
max. nesting of blocks and loops: 10 each,  
max. number of characters per name: 20,  
max. number of bits in bitstrings: 32,  
max. number of characters in strings: 127.

RTL/2

There are a number of restrictions most of which are unlikely to affect the user and are difficult to express in terms of source text limitations. The more significant limitations are:

Item	Configuration	
	Minimum	Recommended
no. of identifier names	180	250
no. of constants other than integers/bytes in range 0-255	200	400
no. of bytes in strings	1000	3000

The following limitations apply to all existing compilers:

Names : maximum of 31 characters  
Arrays : maximum dimension 16  
          maximum length 32767  
Blocks : maximum number 255  
          maximum depth  
          of nesting 15

In practical terms the maximum size of program is likely to be 500 lines of source on the minimum configuration and over 1000 lines on the recommended configuration.



11- In terms of statements or lines per minute, state the rate of compilation for:

- the minimum hardware configuration,
- the recommended hardware configuration.

ALGOL 68

In both cases the rate is about 300 lines per minute with listing.

CORAL 66

For minimum hardware configuration:

150 lines/minute disregarding the overhaed time for loading the 3 passes of the compiler from paper-tape.

For recommended hardware configuration:

500 lines/minute.

PAS 1 - PL/I

Question cannot be answered.

PEARL

BBC-Subset:

Question cannot be answered.

ASME-Subset:

It depends on the target machine and the program and is about 40 lines per minute with listing.

PROCOL

(\*)

RTL/2

Rate of compilation on System 4-70 with recommended configuration is about 500 lines per minute.

Rate of compilation on PDP-11-35 with recommended configuration is about 150 lines per minute.

Figures for minimum configuration not available.

12- How many passes are involved in each compilation?

ALGOL 68

Four passes.

CORAL 66

Three, to enable compiler to run on minimum specified configuration.

PAS 1 - PL/I

8 passes for compilation of PL/I-Subset programs

3 passes for compilation of PAS 1-programs

2 passes for linkage of PL/I and PAS 1-programs.

PEARL

BBC-Subset:

Ten passes.

ASME-Subset:

For the machine independent part: 9 passes plus 4 passes for the system part compilation.

For the machine dependent part: It depends on the minimum hardware configuration. 1-3 passes for code generation.

PROCOL

T 2000 : 3 passes

MITRA 15 : 4 passes.

RTL/2

On S4/360/370

7 passes

On PDP 11

7 passes.

13- State media on which compiler passes are held.

ALGOL 68

Disk (although they could be held on any sequential medium).

CORAL 66

Disk.

PAS 1 - PL/I

Disk-files.

PEARL

BBC-Subset: disk-files.

ASME-Subset: disk-files.

PROCOL

T 2000 : on disk

MITRA 15: on disk.

RTL/2

S4/360/370 : disk

PDP 11 : disk/papertape

14- State media on which intermediate code is held.

ALGOL 68

Disk.

CORAL 66

Disk.

PAS 1 - PL/I

Disk-files.

PEARL

BBC-Subset: Disk-files

ASME - Subset: Disk-files.

PROCOL

T 2000 : On disk  
MITRA 15: On disk.

RTL/2

S4/360/370: core  
PDP 11: core/disk.

15- Is batch compilation of several first passes, then the second passes, etc., possible?

ALGOL 68

No.

CORAL 66

Yes.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No  
ASME-Subset: No.

PROCOL

No for both implementations.

RTL/2

No for both implementations.

16- What is the minimum overlay area?

ALGOL 68

17k (24 bit words).

CORAL 66

16k words.

PAS 1 - PL/I

512k words.

PEARL

BBC-Subset:

Minimum overlay area is one module containing one procedure.

ASME-Subset:

8k (minimum of 24 bit words).

PROCOL

(\*)

RTL/2

S4/360/370: 20k bytes

PDP 11 : 11k bytes.

17- What was the cost of the compiler in man years and how was it written (by a small team, large team, individuals...)?

ALGOL 68

Approximately 6 man years for the machine independent part and 2 man years for the bootstrap.

Written by: Small team (4 persons).

CORAL 66

Cost: 18 man-months

Written by: one individual.

PAS 1 - PL/I

Cost: Not ascertainable

Written by: Small team (5 persons).



PEARL

BBC-Subset:

Cost: About 7 man years

Written by: Small team ( 4 persons).

ASME-Subset:

For the machine independent part a team of five men worked continuously one year together.

For the machine dependent part SIEMENS 306 (AEG 60-50):

Cost for code generation 1 man year (1)

Cost for run-time package (including O.S. extensions) 6 man year.

For code generation 1 man (1)

For run-time package 4 men (4).

PROCOL

Cost: About 4,5 man years

Written by: Small team.

RTL/2

The front-end of the compiler took about 1 man year and each back end slightly less than 1 man year for the initial development.

Subsequent revision has added about 50% to these figures. The front end and each back-end were written by individuals.

18- Was it possible to use any special aids in the construction of the compiler, such as:

- Syntax analyser generator?
- Syntax improving device?
- Other techniques?

ALGOL 68

- Syntax analyser generator:

Yes, PSYCHO was used for the writing of the first syntax analyser.

- Syntax improving device:

No.

ALGOL 68 (continued)

- Other techniques:

The compiler was bootstrapped up from a primitive subset (using PSYCHO) and is now written completely in itself. The third pass of the compiler generates a machine independent code (ZCODE) therefore giving the compiler complete machine independence.

CORAL 66

- Syntax analyser generator:

Yes.

- Syntax improving device:

Yes, in the sense that the syntax analyser also one-tracks the syntax.

- Other techniques:

Bootstrapping from another machine.

PAS 1 - PL/I

- Syntax analyser generator:

No.

- Syntax improving device:

No.

- Other techniques:

The compilers for PAS 1 and PL/I-Subset are written in the PL/I-subset so that they can be translated by the PL/I-Subset-compiler itself into machine code for the Honeywell 316/516.

PEARL

BBC-Subset:

- Syntax analyser generator:

No.

- Syntax improved device:

No.

- Other techniques:

The compiler is written in a PL/I-subset which is to the greatest possible extent contained in the BBC-PEARL-SUBSET.

PEARL (continued)

ASME-Subset:

- Syntax analyser generator:

No. For the table driven syntax analyser a special table generator and special language (similar to BNF Notation) was developed.

- Syntax improved device:

No.

- Other techniques:

To meet the requirements of extensability and portability following techniques are used:

- . Syntax analyser generator
- . portable programs are written in Fortran (ISO-Standard-1)
- . the last pass of the front-end of the compiler generates the machine independent code CIMIC.
- . The macro processor STAGE 2 for code generation.

PROCOL

- Syntax analyser generator:

No.

- Syntax improved device:

No.

- Other techniques:

Several intermediate languages are produced as interface between passes.

RTL/2

Various table driven routines are used, particularly for expression analysis and optimization.

19- Which language features facilitated the use of any special aid?

ALGOL 68

None.

CORAL 66

The system was intended to be one-trackable to allow the use of the syntax analyser generator, SID. See Computer Journal, May 1968, Foster.

PAS 1 - PL/I

The possibilities of PL/I.

PEARL

BBC-Subset: The possibilities of PL/I.

ASME-Subset: None.

PROCOL

None:

RTL/2

None.

20- Does the compiler process macros?

ALGOL 68

No.

CORAL 66

Yes.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No.

ASME-Subset: No.

PROCOL

No.

RTL/2

No. Any macro processing is quite independent of the compiler.

21- Is the macro facility an addition to the basic language?

ALGOL 68

Not relevant.

CORAL 66

No.

PAS 1 - PL/I

Not relevant.

PEARL

BBC-Subset: Not relevant.

ASME-Subset: Not relevant.

PROCOL

Not relevant.

RTL/2

Not relevant.

22- Specify the character set used.

ALGOL 68

The character set used in the 4130 implementation is as follows:

A-Z Ø-9 space line-feed tabulate.

1/2 comment character (in place of #)



ALGOL 68 (continued)

\$	format symbol	
'	stop word delimiter	
()	parantheses	
%*+< / >	operator symbols	
,.:;	punctuation	
10	exponent symbol	
@	at	
[ ]	brackets	
£	used as <u>then</u> etc.	( in place of   )
"	stop quote	
↑	exponentiation symbol	

CORAL 66

ISO8 without lower-case letters (ICL graphic set)

PAS 1 - PL/I

48- and 64-character set of PL/I.

PEARL

BBC-Subset: 64-character set of PL/I.

ASME-Subset: letters: A-Z  
digits : 0-9  
special characters: BLANK + - x / ( ) , . ' : ; =

PROCOL

See Algorithmic Question main: A-Z 0-9 ( ) ' % + - \* / : ;  
! , = # < > \$ space

RTL/2

RTL/2 programs are written in a very specific subset of ISO7 as described in the specification manual.

23- Describe the language representation for the used compiler.

ALGOL 68

The only important points to note are that strop (underlined) words are delimited by the single quote and that £ is used as the representation for | .

CORAL 66

Language symbols are enclosed in single quotes and symbols as assignment and comparators are constructed from multiple characters as necessary.

PAS 1 - PL/I

See 18. (\*)

PEARL

BBC-Subset: See 18 (\*)

ASME-Subset: Not printable characters are replaced by others or composed by overprinting.

PROCOL

PROCCL (\*)

RTL/2

These compilers conform precisely to the specification.

24- Describe program layout.

ALGOL 68

Completely free format. However note that a line-feed acts in the same way as a space and therefore will terminate an identifier.

CORAL 66

Except in strings, layout characters are ignored by the compiler, but it is normal to have one statement per line with indentation reflecting the block structure.

PAS 1 - PL/I

Free format.

PEARL

BBC-Subset: Free format.

ASME-Subset: Program layout is of free format. The first 72 characters of a punch card are processed, all 80 characters are documented.

PROCOL

Free format.

RTL/2

Program layout is basically free format.

25- Describe the comment facility.

ALGOL 68

'CO' to 'CO'

'COMMENT' to 'COMMENT'

1/2 to 1/2 (no # available).

CORAL 66

Comment is allowed between the delimiters "COMMENT" and ";" wherever a declaration or statement is permitted, between round brackets after any semicolon in the program, or after "END" if it takes the form of an identifier.

PAS 1 - PL/I

/\* comment \*/

PEARL

BBC-Subset: /\* comment \*/

ASME-Subset: /\* comment \*/ excluding this two characters  
sets possible within or between statements.

PROCOL

Possible within or between statements, enclosed by "!"

RTL/2

Comments are inserted between % characters but must be all on one line.

26- Describe any additional documentation features.

ALGOL 68

None- excepting the self-documenting nature of ALGOL-style languages.

CORAL 66

None.

PAS 1 - PL/I

None.

PEARL

BBC-Subset: None.

ASME-Subset: None from the compiler.

PROCOL

None.

RTL/2

The TITLE statement enables the source and object code to be labelled in whole or in part.

27- What form of an object code is output?

For example: Assembly language, relocatable binary, absolute binary).

ALGOL 68

Two forms of object code are output:

- (a) Machine independent Assembly Language after pass 3.
- (b) Relocatable binary after pass 4.

CORAL 66

Relocatable binary.

PAS 1 - PL/I

- (a) Machine independent intermediate language
- (b) relocatable binary code
- (c) absolute binary code (produced by the linker).

PEARL

BBC-Subset: (a) machine independent intermediate language  
(b) PDP 11 assembler code.  
ASME-Subset: Assembly language.

PROCOL

Assembly language.

RTL/2

Assembly language.

28- Does the implementation allow machine level code?

ALGOL 68

Yes, between the 'CODE'....'EDOC' construction.

CORAL 66

Yes.

PAS 1 - PL/I

No.



PEARL

BBC-Subset: No.

ASME-Subset: No.

PROCOL

Yes.

RTL/2

Yes.

29- Does the user have to describe his shared-core overlay scheme  
in the source language?

ALGOL 68

No.

CORAL 66

Yes.

PAS 1 - PL/I

Not in the source language but by linker commands.

PEARL

BBC-Subset: Yes, there is one main module in which the overlay  
structure must be described.

ASME-Subset: Yes, by linker commands.

PROCOL

No, this is handled by the Editor.

RTL/2

No. This is a linker function.

30- State the rate at which the compiler produces object code instructions from source text, allowing for processing time through each pass of the compiler. (This question should be answered for the minimum and recommended hardware configurations).

ALGOL 68

About 300 lines of source producing 1,5-2k of code (24 bit words) per minute.

CORAL 66

Minimum:

500 instructions/minute, ignoring the time taken to load the 3 passes of the compiler from paper tape.

Recommended:

1500 instructions/minute.

PAS 1 - PL/I

Not ascertainable.

PEARL

BBC-Subset:

Strongly dependent on output, diagnostics, program structure and host computer.

ASME-Subset:

About 40 lines of source producing 1-1,5k (words) of code per minute.

PROCOL

Not yet performed.

(\*)

RTL/2

See answer to 11.

31- State the time to compile a null program. (This question is designed to ascertain the overhead on compile time.)  
(The question should be answered for the minimum and recommended hardware configurations.)

ALGOL 68

About 8 seconds, mainly taken up in reading compiler tables and information about the environment in which the program is being compiled.

CORAL 66

Minimum: 15 secs.

Recommended: 7 secs. (\*)

PAS 1 - PL/I

Not ascertainable.

PEARL

BBC-Subset: 1 sec on IBM 370/158

ASME-Subset: About 10 seconds.

PROCOL

Not yet performed. (\*)

RTL/2

Time for null program (seconds) :

	configuration	
	minimum	recommended
on S4/360/370	?	7
on PDP 11	?	10

Linkage Editing

32- Can the object code of the compiler be linked with the object code of the object machine assembler?

ALGOL 68

ICL 4100 : No, although in principle yes.

IBM 370 : Yes.

CORAL 66

Yes.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No

ASME-Subset: Yes.

PROCOL

Yes (Editor).

RTL/2

Yes.

33- Can the object code of the compiler be linked with that of other languages? If yes, state which.

ALGOL 68

ICL 4100 : No.

IBM 370 : Probably BCPL, but not IBM compiler output.

CORAL 66

Yes, Fortran IV.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No.

ASME-Subset: SIEMENS 306 : Yes, with Fortran IV procedures.

AEG 60-50 : No.

PROCOL

Yes; there exist means to link PROCOL object code with others object code decks (FORTRAN, PL 15, BASIC, etc....)

RTL/2

Yes. A set of macros exist which enable FORTRAN subroutines to be called from RTL/2 on System 4/360/370.

34- Is a symbol table produced that can be used by a symbolic debugging aid? If so,

- Does it contain external names only?
- Can it include internal identifiers for individual segments (i.e. not the whole system)?

ALGOL 68

Not directly.

- External names are kept in a form not related to the identifiers used by the programmer. All procedure names are present in the machine independent code produced by pass 3 and in the 4130 implementation are left in the machine code for debugging.
- No.

CORAL 66

- Optionally, yes.
- Yes.



PAS 1 - PL/I

- Only for static variables and variables declared as accessible by the operator at runtime.
- No.

PEARL

BBC-Subset:

- Yes, it is produced.
- No, it contains all identifiers.
- Yes.

ASME-Subset:

SIEMENS 306: 'global' names can be accessed at runtime by special debugging facilities, since the user-defined symbols can be connected with the respective memory location at runtime.

AEG 60/50: None.

PROCOL

No.

RTL/2

No.

35- Do facilities exist which allow the linkage editor to omit debugging facilities inserted by the compiler or is recompilation necessary?

ALGOL 68

On the ICL 4130 implementation recompilation would be necessary.

CORAL 66

This particular compiler does not insert debugging facilities.

PAS 1 - PL/I

Recompilation necessary.

PEARL

BBC-Subset:

Recompilation necessary.

ASME-Subset:

SIEMENS 306: Another code generation pass must be performed.

AEG 60/50: Recompilation is necessary.

PROCOL

No.

RTL/2

No. Recompilation is necessary.

36- Does the linkage editor operate in real time?

ALGOL 68

No.

CORAL 66

Yes.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No.

ASME-Subset:

SIEMENS 306: No, the standard linkage editor of the vendor is used.

AEG 60/50: No.

PROCOL

No, the standard linkage editor of the MITRA 15 is used.

RTL/2

Depends on operating system.

37- Minimum recommended configuration for linkage editor in terms

- of: - Core  
- Disk  
- Peripherals.

ALGOL 68

ICL 4130: Linkage is done in ZCODE and the machine facilities required are the same as for the rest of the compiler.

IBM 370 : Linkage is done in object modules. the machine requirements are dependent on the level of LE used.

CORAL 66

- 16 or 24k words depending on the Operating System used.
- Disk with two character streams.
- Teletype.

PAS 1 - PL/I

Same as for the compiler. (See 9).

PEARL

BBC-Subset: Linkage is done on the PDP-11 by the linker LINK11 of DEC.

ASME-Subset:

Item	Siemens 306	AEG 60/50
Core	4,5k words (24 bit)	2
Disk	150k words (24 bit) including library funct.	-
Peripherals	Typewriter, disk (or drum)	Typewriter, disk

PROCOL

See manufacturer's documentation: 4029 P2 / EN  
4399 P / EN from cii

RTL/2

To link for System 4/360/370 - see manufacturer's documentation.

To link for PDP-11.

	On PDP-11	On System 4/360/370
Core	8k	128k
Disk	-	1 disk
Peripherals	paper tape P/P	something

38- What is the capacity of the linkage editor in terms of identifiers?

ALGOL 68

ICL 4130: 3000 - but it can be changed by parameters. This is sufficient for the compiler itself.

IBM 370 : Dependent on LE used.

CORAL 66

Over 3000.

PAS 1 - PL/I

Not ascertainable (some 100 identifiers).

PEARL

BBC-Subset: Limited only by disk space.

ASME-Subset:

SIEMENS 306: Only limited by disk- (drum-) storage.

AEG 60/50: ?

PROCOL

Only limited by disk storage.

RTL/2

To link for System 4/360/370 - see manufacturer's documentation.

To link for PDP-11.

150 each of external and entry named bricks.

39- What overlay facilities are provided by the linkage editor?

ALGOL 68

ICL 4130: Simple one-level overlays.

IBM 370 : Standard OS LE facilities.

CORAL 66

None.

PAS 1 - PL/I

One-level overlay.

PEARL

BBC-Subset: Multiple level overlays.

ASME-Subset:

SIEMENS 306: There is an existing additional overlay-linkage-editor, which may link procedures to a task in a choosable manner.

AEG 60/50: None.

PROCOL

A special command exists (TREE).

RTL/2

To link for System 4/360/370 - see manufacturer's documentation

To link for PDP-11. The normal tree overlay mechanism is provided with optional AUTOMATIC/MANUAL/SYNCHROMESH control.

Additional structure is imposed to cater for memory management on large systems.

40- Is control over these overlay facilities accessible to the user or are the control codes output in the object code by the compiler?



ALGOL 68

ICL 4130: No. They are only used for making up a compiler.

IBM 370 : Yes, as standard LE commands.

CORAL 66

N/A. (\*)

PAS 1 - PL/I

Overlays are user defined by linker commands.

PEARL

BBC-Subset: Control is given to the user.

ASME-Subset:

SIEMENS 306 : The user must specify the segment tree.

AEG 60/50 : No.

PROCOL

Yes, accessible to the user.

RTL/2

See answer to 39.

Run Time

41- What is the minimum, run time, hardware configuration in terms of: - Core store?

- Backing store?

- Business peripherals?

ALGOL 68

- Core store: About 10k (excluding any operating system components).

- Backing store: None.

- Business peripherals: Card reader (if primary input needed) and lineprinter.

AD-A032 571

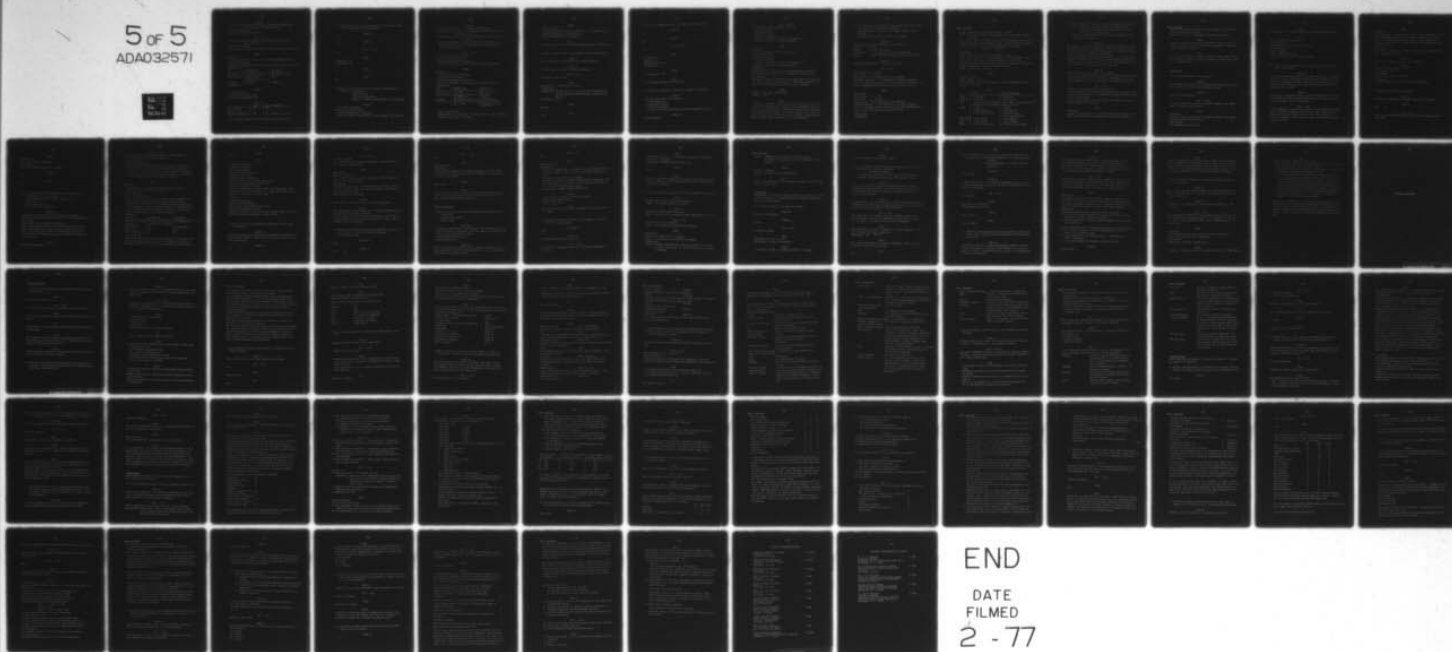
PURDUE UNIV LAFAYETTE IND PURDUE LAB FOR APPLIED IND--ETC F/G 9/2  
A LANGUAGE COMPARISON. A COMPARISON OF THE PROPERTIES OF THE PR--ETC(U)  
OCT 76 R ROESSLER, K SCHENK

N00014-76-C-0732

UNCLASSIFIED

NL

5 of 5  
ADA032571



END

DATE  
FILMED  
2 - 77

CORAL 66

- Core store: 1k words for loader plus whatever Operating System facilities the program requires.
- Backing store: None.
- Business peripherals: Teletype.

PAS 1 - PL/I

- Core store: Minimum defined by minimum operating system (about 6k words ).
- Backing store: None.
- Business peripherals: One input and one output device (e.g. ASR).

PEARL

BBC-Subset:

- Core store: Minimum defined by the minimum operating system.
- Backing store: None.
- Business peripherals: One input and one output device (e.g. ASR).

ASME-Subset:

Item	SIEMENS 3Ø6	AEG 6Ø/5Ø
Core store	32k words (24 bit)	12k words (24 bit)
Backing store	disk or drum (needed by the O.S.)	disk
Business peripherals	typewriter	typewriter

PROCOL

- Core store: 32k
- Backing store: Disk + TTY
- Business peripherals: None.

	RTL/2	
	PDP-11	System 4/36Ø/37Ø
Core (bytes)	4k *	64k
Backing store	-	-
Business peripherals	TTY	Card reader/line printer

\* only really need about 1k but cannot buy them that small.

42- Does the minimum, run time, hardware configuration exceed the minimum, hardware configuration?

ALGOL 68

No.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No.

ASME-Subset: No.

PROCOL

No.

RTL/2

No.

43- What is the maximum run time, hardware configuration in terms of: - Core store?

- Backing store?

- Business peripherals?

- Special peripherals addressable in the language?

ALGOL 68

- Core store: Unlimited.

- Backing store: Unlimited.

- Business peripherals: Unlimited.

- Special peripherals addressable in the language: Not relevant.

CORAL 66

- Core store: 56k words ( a hardware design limit).
- Backing store: At least 32 exchangeable disk drives or magnetic tape units.
- Business peripherals: With multiplexing there is virtually no limit.
- Special peripherals: As any peripheral can be accessed via macros and code or special procedures any peripheral available can be accessed.

PAS 1 - PL/I

- Core store: 32k 16 bit words.
- Backing store: Unlimited.
- Business peripherals: Unlimited.
- Special peripherals: All BBC process peripheral devices.

PEARL

BBC-Subset:

- Core store: 124k 16 bit words.
- Backing store: Unlimited.
- Business peripherals: Unlimited.
- Special peripherals: All BBC process peripheral devices.

ASME-Subset:

Item	SIEMENS 306	AEG 60/50
Core store	64k words	19k (24 bit)
Backing store	each disk is capable of 8000k words	512k words (24 bit)
Business peripherals	10 channels for slow devices, 5 channels for fast (DMA)	13 channels for digital and analog I/O and 32 interrupt connections

- Special peripherals:

SIEMENS 306: It is possible to address the full range of CAMAC-addresses in terms of language.



PROCOL

- Core store: 32k
- Backing store: Several disk, several etc. ( no restrictions).
- Business peripherals: Not relevant.
- Special peripherals: No restrictions.

RTL/2

No restrictions apart from those imposed by machine architecture.

44- What is the character set available to run time programs?

ALGOL 68

The standard ICL 4130 character set (modified ASCII).

CORAL 66

No restriction ( depends on Operating System).

PAS 1 - PL/I

64 character set of PL/I.

PEARL

BBC-Subset: 64 character set of PL/I.

ASME-Subset:

SIEMENS 306: The same set as in 22. There are some additional special characters whose external representation may vary.

AEG 60/50: see 22.

PROCOL

TTY set.

RTL/2

ISO.

45- Does the language restrict the character set available?

ALGOL 68

No.

CORAL 66

No.

PAS 1 - PL/I

No.

PEARL

BBC-Subset: No.

ASME-Subset:

SIEMENS 306: see 44.

AEG 60/50: No.

PROCOL

No restriction (TTY).

RTL/2

The language imposes no restrictions.

46- What run time support software is needed? Give details.

ALGOL 68

The following items are necessary:

- A small executive
- A diagnostic package
- A basic I/O package
- A general character handling I/O routine supplied by the O.S..

CORAL 66

None essential.

PAS 1 - PL/I

The run time package consists mainly of:

- kernel executive
- I/O device drivers
- diagnostic package
- package for character handling (formatted I/O)
- package for handling operator commands.

PEARL

BBC-Subset:

The run time package consists mainly of:

- kernel executive
- I/O device drivers
- diagnostic package
- package for character handling ( formatted I/O)
- package for handling operator commands.

ASME-Subset:

SIEMENS 306: The vendors O.S. and some additional features:

Facilities for tasking-timing, standard-I/O, graphic-I/O, file-handling.

AEG 60/50: A widely modified O.S. and some additional features for: Task activation and terminating, standard procedures, linear editing, simulating of floating point arithmetic.

PROCOL

MITRA 15: The vendor 'MMT' monitor.

T 2000 : None.

RTL/2

A package is needed to initialize the stack for each task - this is only a few instructions. There is also a package of routines to carry out such actions as procedure entry and exit (with monitoring where applicable), array bound checks, etc.. In cases where floating point hardware is not available, a software package is needed if REAL arithmetic is to be performed.

47- Give details of the method of storage and precision of each data type of the language. For example:

Floating point: 1 bit sign 7 bit signed exponent 24 bit mantissa;

Logical: true or false held in an 8 bit-byte.

#### ALGOL 68

Floating point: 39 bit mantissa, 9 bit exponent

Integer : 24 bit

Boolean : 24 bit (0  $\equiv$  false)

Characters : 7 bit character held in 24 bit word

String : 46 bit characters packed in every 24 bit word .

#### CORAL 66

INTEGER and FIXED both in stored 16 bit word.

#### PAS 1 - PL/I

Binary fixed : 1 - 30 bit

bit string : 1 - 32 bit

binary float : 21 bit mantissa, 8 bit exponent

character string: 1 - 127 characters, 8 bit per character.

Arrays of type binary fixed, bit and character may have the attribute UNALIGNED to pack more than one item into one word.

#### PEARL

BBC-Subset:

Binary fixed : 1 - 31 bit

bit string : 1 - 32 bit

binary float : 24 bit mantissa, 7 bit exponent

character string: 1 - 127 characters, 8 bit per character.

Array of the binary fixed and bit may have the UNALIGNED attribute to pack more than one item into one word.

ASME-Subset:

SIEMENS 306:

PEARL (continued)

Fixed: 1 bit sign, 23 bit signed number (1 word)

Float: 1 bit sign, 10 bit signed exponent, 33 bit signed mantissa  
(2 words)

Bit strings: Held in one word each (therefore maximum length is 24)

Character string: Maximum length is 40, one word holds 4 Characters

Clock and Duration: Held in one computer word as integer  
(precision is sec/10).

AEG 60/50:

Fixed: 1 bit sign, 23 bit signed number (1 word)

Float: 1 bit sign, 6 bit signed exponent, 17 bit mantissa (1 word)

Bit string: 1 word also for logical false '1' and true '0'

Character string: Maximum length is 40, one word holds 4 characters

Clock and Duration: 1 word (precision is sec/1)

Label: 1 word (within array).

PROCOL

Integer: 16 bits

Double integer: 32 bits

Floating point: sign 1 bit, exponent: 7 bits, mantissa: 24 bits

Logical: 1 to 16 bit

RTL/2

	PDP-11	System 4/360/370
BYTE	8 bit byte	8 bit byte
INT/FAC	16 bit word	32 bit word
REAL	1 bit sign, 7 bit binary exponent, 24 bit mantissa	1 bit sign, 7 bit hex exponent, 56 bit mantissa
LABEL	2 words 1 word address 1 word level pointer	3 words 1 word address 1 word proc base 1 word level pointer
PROC/STACK	1 word address	1 word address
REF	1 word address	1 word address
ARRAY	1 word length prefix	half word length prefix



48- Give evidence of run time storage efficiency when compared with:

- Assembly language coding for target machine.

- Object code compiled from other source languages for the same target machine.

(Note, a standard bench mark test would be needed to make the answers comparable).

#### ALGOL 68

Any answer to this question must be dependent upon the particular language construct examined and could vary wildly depending on which construct is chosen. For example, on the 4130 addressing of non-local quantities is expensive.

Assembly language coding for target machine: 2 - 10 times worse  
Object code compiled from other source languages for the same target machine: We estimate that it is 25 - 50% worse than code from the ALGOL 60 compiler on the same machine.

#### CORAL 66

Assembly language coding for target machine:

Estimate uses 10 per cent more than good assembly language coding.

Object code compiled from other source languages for the same

target machine: Estimate uses half the core used by Fortran IV.

#### PAS 1 - PL/I

Assembly language coding for target machine:

A comparison of an application package including a tailored operating system resulted in about 20% more storage than the assembler written package.

Object code compiled from other source languages for the same target machine: Not relevant.

#### PEARL

BBC-Subset:

Assembly language coding for target machine: Data not yet available but estimation is approximately the same as for PAS 1.

PEARL (continued)

Object code compiled from other source languages for the same target machine: Not relevant.

ASME-Subset: Not yet performed for the target machines.

PROCOL

Assembly language coding for target machine: About 1,4 worse.

Object code compiled from other source languages for the same target machine: 1,8 better.

RTL/2

This is very difficult to answer. Figures of 20% less efficient than assembly language and 40% more efficient than other languages have been quoted.

Diagnostics

49- Describe the compile time listing.

ALGOL 68

Line numbers and source text. The machine code generator outputs details about the size of object code generated.

CORAL 66

None with this compiler.

PAS 1 - PL/I

Listing of source program with statement number, level number and nest number (like IBM-PL/I).

PEARL

BBC-Subset:

Listing of source program with statement number, level number and nest number (like IBM-PL/I).

ASME-Subset:

Line numbers and source text.

#### PROCOL

A description is too long; seeing the listing should be better.

#### RTL/2

The following can be obtained:

Source listing

Concordance give use of all identifiers

Warning messages

Object listing

Use of compiler resources

Summary of brick sizes in object program.

50- What are the diagnostics (e.g. error lists, store map, object code listing)?

#### ALGOL 68

Syntax error messages appear in the text. Pass 2 and 3 messages appear at the end. The machine code generator can print the code produced in Assembly Language mnemonics and will summarize the code produced at the end. Full compiler tracing and inter pass dumps can be requested.

#### CORAL 66

Each error is described briefly, with the page and line on which it occurred, and the preceding 60 characters. Optional store-map can be obtained from the Link Editor.

#### PAS 1 - PL/I

Error diagnostic, cross reference table of all identifiers, store map, intermediate language code listing, object code listing, details on program modules (entry points, static segments,...), overall information (total storage, number of procedures, tasks,...), compiler tracing.

PEARL

BBC-Subset:

Error diagnostic, cross reference table of all identifiers, store map, intermediate language code listing, object code listing, details on program modules (entry points, static segments,...), overall information (total storage, number of procedures, tasks,...), compiler tracing.

ASME-Subset:

Error listing, reference listing of source names to intermediate code, object code listing.

PROCOL

Errors list, store map, object code (optional)

RTL/2

For each error a warning is given:

Line number in source

Error number

Current symbol and/or identifier

A message

The message may be omitted in small configurations.

51- Which options can be suppressed?

ALGOL 68

Listing, cross reference, compiler tracing, code listing.

CORAL 66

N/A (\*)

PAS 1 - PL/I

Every option except source listing and error diagnostic may be suppressed.

PEARL

Subset:

option may be suppressed.

-Subset:

reference listing, object code listing.

PROCOL

RTL/2

Indicate which run time diagnostics are available from:

- Arithmetic overflow reporting,
- Bound checking (Arrays, switch lists, etc.)
- Label trace
- Block entry counts
- Other facilities (specify).

ALGOL 68

Arithmetic overflow reporting: floating point overflow.

Bound checking (Arrays, switch lists, etc.): Array bounds are checked.

Label trace: Zcode labels can be traced.

Block entry counts: Procedure entry and exit can be traced.

Other facilities: Stack, heap, last 32 procedures calls and its, file status and others can be checked. All items can be turned off either dynamically or by parameter to the diagnostic routine.

CORAL 66

with this compiler.



PAS 1 - PL/I

- Arithmetic overflow reporting: Overflow message optional.
- Bound checking: Bound checking optional.
- Label trace: Yes.
- Block entry counts: Yes.
- Other facilities: Generalized trace facilities, interrogation and alteration of every storage location via its address, interrogation and alteration of variables declared accessible via their names, starting of "operator procedures", underflow checking optional, zero divide checking optional.

PEARL

BBC-Subset:

- Arithmetic overflow reporting: Overflow message optional.
- Bound checking: Bound checking optional.
- Label trace: Yes.
- Block entry counts: Yes.
- Other facilities: Generalized trace facilities, interrogation and alteration of every storage location via its address, interrogation and alteration of variables declared accessible via their names, starting "operator procedures", underflow checking optional, zero divide checking optional, full range of tasking statements with all possible schedules.

ASME-Subset:

Question	SIEMENS 306	AEG 60/50
Arithmetic overflow reporting	No (but can easily be provided if necessary)	in preparation
Bound checking	Yes	in preparation
Label trace	Yes	No
Block entry counts	No	No

Other facilities:

SIEMENS 306: Source line tracing is optional (in connection with other error messages like I/O errors, formatting errors, etc.)

AEG 60/50: errors from the O.S. are documented on the typewriter.

PROCOL

None.

RTL/2

On System 4/360/370:

Arithmetic overflow

Array bound checking

Generalized GOTO checking

Procedure entry count

Procedure entry retrotrace

Procedure entry trace - with local values

Stack overflow and maximum usage check

Current line number in source

Diagnostic package prints out current nested procedure calls with values of local variables, etc.. User can provide his own package - written in RTL/2 of course.

On PDP-11:

Array bound checking

Generalized GOTO checking

Stack overflow and maximum usage check

Current line number in source

Diagnostic package halts machine and displays reason for error.

User can provide own package - written in PAL.

53- How do the run time diagnostics affect the run time code efficiency?

ALGOL 68

The major overheads are checking stack bounds and recording procedure entry and exit. Running with full tracing slows the program by about 20%.

CORAL 66

N/A

(\*)

PAS 1 - PL/I

Slowing by checks.

Any slowing by tracing possible dependent on used facilities  
(e.g. step by step tracing).

PEARL

BBC-Subset:

Slowing by checks.

Any slowing by tracing possible dependent on used facilities  
(e.g. step by step tracing).

ASME-Subset:

SIEMENS 306: Label trace and source-line trace affect the run  
time efficiency by using two additional machine instructions  
for each label or line.

AEG 60/50: Not yet performed.

PROCOL

Not relevant, since there are no run time diagnostics.

RTL/2

All diagnostics can be switched off if really necessary and pro-  
gram then has no residual overhead.

If full diagnostics in use then a pathological case might run  
at half speed (e.g. null procedure called in a loop). Typical  
overhead only a few percent.

54- Specify any language feature which hindered or assisted in  
the inclusion of any of the above diagnostics.  
State which and how.

ALGOL 68

None.

CORAL 66

N/A (\*)

PAS 1 \_ PL/I

None.

PEARL

BBC-Subset: None.

ASME-Subset:

SIEMENS 3Ø6/AEG 6Ø/5Ø: The formatless language-structure makes it difficult to identify the source-line number at machine-code level for line-tracing.

PROCOL

Not relevant.

RTL/2

The fact that all local variables are scalars makes the analysis of the run time stack during error recovery easier than if dynamic arrays had been present as well.

Library Facilities

55- What library facilities are included with the language from:

- Procedures?
- Hardware descriptions?
- System names?

ALGOL 68

- Procedures: Procedures or any section of code between begin...end may be included in sources as a library item or as an object module from an object module library.
- Hardware descriptions: None.
- System names: None.

CORAL 66

There is no specific language library, but any program may call any particular library of simple variables, arrays and procedures that exists in the installation.

PAS 1 - PL/I

None.

PEARL

BBC-Subset: None.

ASME-Subset:

- Procedures: SIEMENS 306: Procedures may be specified with the attribute 'GLOBAL' and so linked to the object module at LE-time.
- AEG 60/50: Only one module is allowed.
- Hardware descriptions: SIEMENS 306/AEG 60/50: The subset does not provide device variables, so the hardware description is fully known at code-generation time.
- System names: SIEMENS 306/AEG 60/50: None.

PROCOL

- Procedures: CII Scientific library Packages.
- Hardware descriptions: None.
- System names: None.

RTL/2

None. The language is basically independent of particular systems.

56- What other library facilities are accessible to programs written in the language?

ALGOL 68

None.

CORAL 66

As normal CORAL procedures (see 55).

PAS 1 - PL/I

A large set of standard procedures for different purposes is available.



PEARL

BBC-Subset: A large set of standard procedures for different purposes is available.

ASME-Subset: SIEMENS 306/AEG 60/50: None.

PROCOL

None.

RTL/2

There are standard packages for streamed input-output and error recovery. There are also mathematical functions, file handling procedures available.

57- How are library procedures written and added to the library?

ALGOL 68

Written: As standard ALGOL 68C procedures.

Added: By standard system utilities.

CORAL 66

Written: As normal CORAL procedures.

Added: By use of the appropriate CORAL communicator (see 55).

PAS 1 - PL/I

Written: In PL/I or assembler.

Added: Not possible by application programmer.

PEARL

BBC-Subset:

Written: In PEARL, PL/I or assembler.

Added: Not possible by application programmer.

ASME-Subset:

Written: SIEMENS 306/AEG 60/50: Library procedures are usually written in Assembler, but may also be written in PEARL or FORTRAN.

PEARL (continued)

Added: SIEMENS 306: By a vendor's utility program.  
AEG 60/50: By a program specially developed for this purpose.

PROCOL

Written: Assembler.  
Added: See manufacturer's documentation.

RTL/2

Library procedures are just procedures written in RTL/2 and are held in common files.

Portability

Portability is a matter of degree and hence is difficult to quantify from answers to a questionnaire. The following questions may have some relevance.

58- In what language is the compiler written?

ALGOL 68

ALGOL 68 and Assembler.

CORAL 66

Itself.

PAS 1 - PL/I

In BBC-PL/I-subset.

PEARL

BBC-Subset: In PL/I (see 18).  
ASME-Subset: Fortran and Assembler.

PROCOL

In Assembler T 2000, in Assembler MITRA 15, in PROCOL.

RTL/2

All RTL/2 compilers are written in RTL/2.

59- Is the compiler written in separate passes falling into the categories: a) Machine independent

b) Machine dependent

such that the output of a) is the input to b) and being in a standard format identified with the language itself?

ALGOL 68

Yes. The first three are machine independent and are written in ALGOL 68, the last pass is the code generator and is written in Assembler. The interpass communication between passes 1-3 is by an in-core tree.

CORAL 66

To some extent, yes, as the first two passes are mainly machine independent. A true machine independent compiler is being written which will only need separate code generators.

PAS 1 - PL/I

Yes, the first 5 (2) passes of the PL/I subset compiler (the PAS 1-compiler) are machine independent and deliver the input for the last 3 (1) passes which are machine dependent.

PEARL

BBC-Subset: Yes, the first 7 passes are machine independent. The last 3 passes are machine dependent.

ASME-Subset: Yes. (See 12).

PROCOL

Not exactly but passes are machine independent (except the last one). The format is not standard.

RTL/2

Yes.

- 60- If the compiler is not split as in 59, what proportion of the compiler is a) Machine independent (or compiler machine dependent).  
b) Machine dependent ( or target machine dependent).

ALGOL 68

Not relevant.

CORAL 66

In principle most of it is machine independent but the syntax analyser and list-processing are tailored to the particular computer for efficiency.

PAS 1 - PL/I

Not relevant.

PEARL

BBC-Subset: Not relevant.

ASME-Subset: Not relevant.

PROCOL

Not relevant.

RTL/2

Not applicable.

- 61- Identify any bootstrapping, cross compilation or other techniques used in compiler writing which would aid portability, and show how.

ALGOL 68

The compiler produces a machine independent Assembly language (Zcode) from pass 3. To bootstrap the compiler it is only necessary to write a code generator and provide a run time system. (1-2 man years of work).

#### CORAL 66

Bootstrapping has been made easier by the development of SID to generate a syntax analyser for a variety of computers. If the code-generation part of the compiler is kept distinct from the rest, it can be replaced by a simple code-generator for another computer, and again bootstrapping is easy.

#### PAS 1 - PL/I

The compilers can run on every (host-) computer on which a PL/I compiler is available. To produce code for a new target machine the machine dependent passes are to be rewritten. Then the compiler can translate itself to be run on the target machine.

#### PEARL

##### BBC-Subset:

The compilers can run on every (host-) computer on which a PL/I compiler is available. To produce code for a new target machine the machine dependent passes are to be rewritten. Then the compiler can translate itself to be run on the target machine.

##### ASME-Subset:

Machine independent parts. (must be written only once)

- a) one unique version translates the source code into the intermediate form (CIMIC) written in Fortran
- b) For code generation a macro processor (STAGE2) exists on each target machine.

Machine dependent parts: (must be rewritten for each target machine)

- a) an interface package for the not standardized Fortran I/O especially for backing store.
- b) a set of macros for the code generation ( translating of CIMIC in Assembler)
- c) a run time package and standard library functions.

#### PROCOL

Bootstrapping.



#### RTL/2

Since the compilers are written in RTL/2 they can be transferred from one machine to another with ease. A compiler for a new machine would be written in RTL/2 and used initially as a cross-compiler on any convenient machine which supports RTL/2. It can then be self-compiled to run on the new machine if necessary.

62- Provide specific evidence of compiler portability that has been achieved.

#### ALGOL 68

The compiler has been bootstrapped onto the following machines: ATLAS, 360 under both O.S. and MTS, ICL 4130, DEC 11/45 CTL Modular 1, DEC 10.

#### CORAL 66

The Modular One compiler was transferred to System 4 in 3 months.

#### PAS 1 - PL/I

The portability facilities of the compiler could not be profited because the language system PAS 1 - PL/I-Subset is not used for the new target machine of BBC. (Here a PEARL-Subset was implemented).

#### PEARL

BBC-Subset:

The portability facilities depend on the fact that about 3/4 of the compiler are machine independent.

ASME-Subset:

SIEMENS 306, AEG 60/50, SIEMENS 404/3.

#### PROCOL

The MITRA 15 compiler has been produced by use of the T 2000 compiler.

RTL/2

An RTL/2 compiler consists of four parts:

- i) The front-end: There is just one unique version. It translates the source program into intermediate form.
- ii) A back-end: One exists for each target machine. This generates the object code from the intermediate code.
- iii) An interface package: This package has three functions -
  - i) to handle storage of symbol tables, ii) to prepare and present diagnostic messages, iii) to control the storage of the intermediate program. Several versions of each part exist appropriate to the size and use of the compiling computer. Each part may be selected orthogonally to other parts.
- iv) A driver program: one exists for each compiling machine. It controls the overlays, etc. and generally interfaces the compiler with the host operating system.

Thus to move a compiler to a different machine all that is necessary is to select appropriate versions of the various parts and link them together. This has been done very successfully a number of times and no alterations have ever been necessary to items i) to iii) to cater for machine or operating system dependence.

LANGUAGE EXPERIENCE

Language Designer

- 1- State the actual applications envisaged when the language was designed.

CORAL 66

Any real-time application.

PAS 1 - PL/I

All kind of process control and industrial data collection.

PEARL

Middle level language for process and experiment automation purposes.

PL/I

Broad range of data processing, computational and multitasking applications.

PROCOL

The language has been designed for covering general Real Time applications and is specially suited for Process Control applications and communications systems.

RTL/2

Industrial data collection, communication and control systems; both application and system programs.

- 2- What were the principal characteristics felt desirable in a language to satisfy these applications? (For example compact code, rapid implementation).

CORAL 66

Efficient code on a variety of computers; ability to trap most errors at compile-time; one-track predictive syntax for fast compilation.

PAS 1 - PL/I

Programming all problems with a high level language without insertions of assembly code, e.g. scheduling of tasks, description of hardware configuration.

PEARL

- Flexibility (for application)
- easy handling
- easy learning
- portability
- documentation value to improve security.

PL/I

Rich language with many powerful features.

PROCOL

The main desirable features to meet the demand of these sorts of applications are:

- An efficient tasking mechanism,
- Bit handling possibilities,
- Efficient and compressed object code,
- Standardization in handling all input-output problems,
- Ease of use similar to FORTRAN.

RTL/2

Conventional efficient high-level algorithmic features should be available.

Dynamic task creation, synchronization and termination should be facilitated.

Subroutines should be re-entrant so that multitask programming is simplified.



RTL/2 (continued)

It should be suitable for the writing of large parts of operating systems as well as application programs.

No unpredictable timing problems should arise (as might happen with storage controlled by a conventional garbage collector).

The language itself should be independent of particular operating systems and should be able to interface with existing systems.

The notation should be straight forward and easy to type on standard equipment.

Programs should be secure and the language should encourage the early detection of errors at compile time and enable efficient monitoring at run time.

The object code should be efficient in space and not call for extensive run time control routines.

The language should encourage precise and tidy program management.

The language should be as simple and familiar as possible within the above constraints so that training problems are minimized.

Great concern for human interfaces led to especial emphasis on areas such as ease of program preparation, documentation, security, program management and maintenance.

3- What standard languages were drawn from in designing the candidate language?

CORAL 66

Mainly ALGOL 60, but also JOVIAL, CORAL 64 and FORTRAN.

PAS 1 - PL/I

PL/I.

PEARL

ALGOL 68, PL/I.

PL/I

PL/I.

PROCOL

JOVIAL, FORTRAN, ALGOL 60, CORAL 66, ASTRE.

RTL/2

The following standard languages were drawn from:  
ALGOL 60, ALGOL 68, FORTRAN IV, PL/I.

The following languages were also drawn from:

RTL/1	(ICI)
CORAL 66	(RRE)
SML	(ICI)
POP/2	(University of Edinburgh)
IMP	(University of Edinburgh)
TPL/1	(University of Essex)
BCPL	(Cambridge University)
ALGOL W	(Wirth/Hoare)

4- What features were directly adopted from these sources, and why?

CORAL 66

Probably none was adopted without some change.

PAS 1 - PL/I

A subset of PL/I was used.

PEARL

- Syntactical equivalence in the algorithmic part like declaration of data types, procedures, standard-I/O, formats, labels from PL/I and
- loop, conditional clause, orthogonal design, block-structure from ALGOL 68.

PL/I

The whole language.

PROCOL

Bit handling from ASTRE, CORAL.

Algorithmic part and structure from FORTRAN.

Tasking and I/O are completely original.

The all system has been thought towards the best efficiency and the easiest use by applicant programmers.

RTL/2

RTL/2 was virtually redesigned from scratch with a lot of influence from ALGOL 60 and ALGOL 68. The features directly taken from an existing language are:

Lexical style - reserved key-words etc.	POP/2
Integer and real modes	ALGOL 60/68
Variable names	PL/I
Procedure calls and parameter mechanism	POP/2
Conditions	POP/2 and CORAL 66
Comparisons	ALGOL 68
Operators, +, -, etc.	ALGOL 68
Precedence of operators	ALGOL 68
Labelling of statements	ALGOL 60
Assignment statements	ALGOL 60
Return statement	PL/I

5- What features from these sources were used as a basis for development of other features, and why were they modified?

CORAL 66

Block structure and procedures from ALGOL, but static instead of dynamic storage allocation for run time efficiency; COMMON from FORTRAN, extended to include PROCEDURES; data packing and part word manipulation from JOVIAL; etc.

PAS 1 - PL/I

No modification of PL/I features.

PEARL

- I/O statements for process peripherals and graphic-I/O from PL/I.
- Reference concept up to level two, structured values, operator and type declarations from ALGOL 68.

PL/I

Not relevant (see 4).

PROCOL

Any modification has been thought in order to take the best advantages of the target machines as near as possible and to have the most efficient object code.

RTL/2

Syntax of numbers - RTL/1 and FORTRAN  
modified to include fractions mode, more legible binary.

LET - BCPL  
initially very like MANIFEST declaration, but extended to cover other simple replacements since cost of doing was so low.

Declaration style - ALGOL 68  
modified to exclude square brackets since not generally available and to increase legibility.

Bytes - RTL/1  
name changed from char to reflect use as small constants as well as character values.

Fractions - RTL/1 and CORAL  
derived from scaled integers of RTL/1 and CORAL. Modified to be simplest possible fixed point facility with low compiler overhead.

LENGTH operator - ALGOL 68, PL/I  
gives upper bound of array - lower bound known to be 1 - name changed since more appropriate with fixed lower bound.

Mode conversion - ALGOL 68  
to suit RTL/2 style.

RTL/2 (continued)

Logical operators	-	CORAL 66
names changed since did not like originals.		
Blocks	-	ALGOL 60
made more explicit since BEGIN-END no longer needed for compound statement and key-word block emphasises use.		
Switch	-	FORTRAN
modified to suit RTL/2 style.		
Iterative statements	-	ALGOL W
modified key-words.		
Coercions	-	ALGOL 68
only dereferencing and widening. Others avoided since confusing and unnecessary.		

6- What features from these sources were dropped? State whether this was because they were considered to be irrelevant, or because they were replaced.

CORAL 66

The implicit recursion of ALGOL 60, because inefficient at run-time.

PAS 1 - PL/I

Most possibilities of record-I/O,  
Data-directed I/O,  
features necessary for commercial data processing, pointers, structures, storage classes BASED and CONTROLLED, array operations.

PEARL

- No default options by reason of security from PL/I
- no collateral elaboration by and - also - taken and no coercion operations from ALGOL 68.

PL/I

Not relevant (see 4).



#### PROCOL

Those features leading to an inefficient object code, heavy to implement on the machine and not really needed by the user.

#### RTL/2

I will consider in detail only ALGOL 60, ALGOL 68, CORAL 66 and RTL/1. PL/I is too extensive to answer in detail and the others are not so relevant.

ALGOL 60 features dropped:

Boolean	use byte or integer flag .
Dynamic arrays	to reduce run time overhead; can just manage without .
Variable lower bound	to reduce run time overhead of subscript checks; unnecessary .
Own	data bricks give static data .
Separate square and round brackets	utility outweighed by lack of availability .
Procedure nesting	to reduce run time overheads; can just manage re-entrancy without unnecessary.

Parameters by name	unified assignment mechanism used for all parameters.
--------------------	---

ALGOL 68 features dropped (in addition to those dropped from ALGOL 60) - main features:

Sizes	unnecessary.
Unions	not good enough - see above.
Trimming	to reduce run time overhead in handling arrays.
Slicing all ways	to reduce run time overhead in handling ar..
Flexible arrays	to reduce run time overhead in handling ar..
Parallel clauses	style disliked because does not emphasize underlying structure adequately - replaced by tasking actions of operating system.

RTL/2 (continued)

Case clause	syntax not liked - comma too weak as separator of actions and no alternative notation emerged - use switch statement instead.
Operator declarations	unnecessary - their use can increase problem of program maintenance by encouraging strange programming styles.
Identity definitions (of plain values)	use LET instead.
Heap	to reduce run time overhead.
Transput	too elaborate for small systems so left outside language.
Many other features of ALGOL 68 are dropped as a consequence - COMPLEX, STRING etc.. It is hoped that enough has been said to give the general flavour.	
RTL/1 features dropped ( other than features of ALGOL 6Ø):	
Section value	this was a program lump - data, task, procedure or stack - indirectly addressed to give language direct ability to describe tasking, storage control etc. - omitted because implies language dependence on operating system.
Task	this was a named section of code - a parallel code element (PCE) - identified now as a parameterless procedure.
Scaled integers	see remark for CORAL 66.
Tasking and I/O	all omitted because too much of a constraint on host systems - can be defined adequately in terms of basic RTL/2 constructs, providing basis for evolution of standards without language modification.

RTL/2 (continued)

CORAL 66 features dropped ( other than features of ALGOL 60):

Table	insecure - effectively replaced by records in most uses.
Overlay	insecure and irrelevant.
LOCATION, anonymous references	insecure - most uses can be replaced by typed references leaving the logically insecure cases in code statements thereby emphasizing their status.
BITS	likely to generate inefficient code on most hardware. Use shifts instead.
Scaled integers	users not interested - fractions introduced as minimal feature.

7- What new features were included in the language and for what purpose?

CORAL 66

Blocks of machine code which can include identifiers currently in scope, to allow access to every machine facility.

PAS 1 - PL/I

Features for hardware description, scheduling, tasking, semaphore variables, interrupt handling, process I/O, operator communication, shift-instructions.

PEARL

- Task concept to meet the requirements for real-time applications.
- Semaphores of Dijkstra-type for synchronization of parallel activities.
- Hardware configuration description on language level for portability.
- Special I/O statements for inter-system communication and a set of statements for graphic data handling.

PEARL (continued)

- File handling statements for secure manipulation of data on mass storage.
- Modularity for separate compilation of modules.
- Special attributes like "reentrant" and "resident" for influencing storage allocation.
- Storage allocation is done implicitly by the block structure of the program, by priority of tasks, by the availability of resources, etc..

PL/I

APG/7 added some experimental process I/O, real time and queue management features to exploit the System/7 hardware.

PROCOL

New features added, to take advantage of particular features of the machine:

- Floating point,
- Library handling,
- Event with delay.

RTL/2

I will include here features not in any of the language of 2.2.1 apart from RTL/1 which is seen as a private prototype.

Strings	ability to insert arbitrary non-printing characters in a unified manner. For deleting characters on strange devices, VDU etc..
Comments	rule regarded newline not allowed - for security of syntax.
Options	general style is unique. For compile time change of options without modification of source.
Bytes	8-bit unsigned integer. For character handling and small integers.



RIL/2 (continued)

Fractions	for cheap fixed point word length independent arithmetic.
Stack	for explicit task control and explicit allocation of re-entrant storage.
Double length	for precise description of intermediate values of expressions, to give maximum accuracy and to simplify integer/fraction conversion.
Shifts-logical	for detailed control in pattern manipulations.
Shifts-arithmetic	for integer and fraction scaling.
External linkage	hardly a new idea but the descriptions of external bricks are not particularly like any other language. Sufficient information is provided to ensure the possibility of full checks at linkage time.
SVC procedure	a special form of procedure call for interfacing with an operating system.
SVC data brick	a data area of which there is a separate private copy for each task. Used in re-entrant I/O and error recovery routines etc..

Compiler Writer

These questions are designed to establish influence of language structure on the compiler.

8- How many compiler passes are necessary for language structure reasons, to produce loadable object code from language source?

CORAL 66

One alone.



PAS 1 - PL/I

- 1 for PAS 1 programs
- 2 for PL/I-subset-programs.

PEARL

PEARL is not qualified for one pass generation.

PL/I

The following questions cannot be answered at this time.

PROCOL

4 passes.

RTL/2

Logically 3 but effectively 2.

- 9- Which features, if any, present difficulty to the compiler writer, and what is the nature of such difficulties?

CORAL 66

Labels, because they alone may be used before declaration; calls of parameterless procedures, because they are difficult to distinguish from simple variables.

PAS 1 - PL/I

Cannot be answered.

PEARL

Modularity, labels, configuration description.

PROCOL

The compiler is in two logical parts:

- Part 1 performs syntactical and semantical analysis (2 passes)
- Part 2 performs the generation for the object machine (2 passes for the MITRA 15 machine).

## RTL/2

The compilers are in two parts - a common front end and a machine dependent back end. We will consider these separately.

### A. FRONT END

There was some difficulty in finding an efficient means of holding mode information, checking modes, widening etc. This was largely due to lack of familiarity with techniques necessary when the number of modes is infinite.

Conditions pose some problems. After IF is not known whether dereferencing will be necessary until the operator has been found to be `:=:`, `:=:` or otherwise. If the first operand is itself conditional and dereferencing is required then for efficiency we wish to force it in each branch of the operand rather than at the node. A quick look ahead is required. The point is that `:=:` and `:=:` are only operators on non-simple modes.

There is a slight confusion between bytes and integers. A number such as 3 is notically always a byte, but compile time conversions to integer mode where appropriate is obviously desirable. The context is not always clear in conditional expressions.

Error recovery in processing declarations needs care when skipping over other text. The key-word PROC is rather overused.

There is a potential confusion in declarations between a repetition factor and a subscript in brackets. The context does, however, make it clear.

### B. BACK END

Some difficulties of the back ends are imposed by the front end and interface philosophy rather than the language.

To my knowledge, compiler writers have had some difficulties with the following language areas:

Double dereferencing implied in GOTO ref label and call of ref procedure.

Storage of intermediate array base address in handling array elements where both the array base and subscript involve array/record access.

10- How did the availability of other languages and software on the machine influence the implementation of the compiler?

CORAL 66

Not at all, because it was bootstrapped from another machine.

PAS 1 - PL/I

Cannot be answered.

PEARL

No influences from other languages are possible.

PROCOL

There was no influence of other languages implemented on the MITRA 15 such as PL 15 or FORTRAN on the design of the PROCOL compiler.

RTL/2

The overall structure of the compiler was influenced by that of the RTL/1 compiler which in turn was influenced by that of the SML compiler written in 1966.

The first RTL/2 compiler was written in RTL/1 and compiled a subset of RTL/2 (multidimensional arrays, fractions and reals were omitted). The production RTL/2 compiler was then written in this subset of RTL/2.

11- Does the structure of the language enable the use of any particular compiler writing techniques designed to speed up compiler preparation? If so, state in what way and with what techniques.

CORAL 66

Yes, the deliberate choice of a one track syntax enabled the use of automatic syntax analyser generators to be made.

PAS 1 - PL/I

Cannot be answered.

PEARL

Language from "Chomsky-type LL(1)", therefore parser generators for LL(1)-grammars are applicable.

PROCOL

Not particularly.

Prime objective have always been run time efficiency.

RTL/2

Not particularly - the language was designed largely with user requirements and run time efficiency as prime objectives. However, the compiler itself has been carefully structured into front end, data base and driver so that the production of new compilers both for new target machines and for running in different environments is simplified and also to ensure that the language accepted by various compilers is as uniform as possible.

Language User

These questions are aimed at, and should be answered for, on-line industrial applications.

12- State application areas in which the language has been used.

CORAL 66

Our particular Modular 1 is a Hands-on system that also run data links and special displays but the language has been used by other people for all conceivable applications.

PAS 1 - PL/I

System programming: the compilers for PAS 1 and PL/I-subset.

Applications: data logger, control of production plant, turbine testing system, laboratory automation, house automation, power distribution.



PEARL

Experiences of language users not yet available.

PROCOL

Chemical production systems; Launching systems; Control systems; R.T. systems, etc..

RTL/2

The answers for this part are based on the results of posing the questions to 27 individual users.

In some cases individual users had used RTL/2 in quite different areas and accordingly provided more than one set of answers, whereas in other cases a group of users had worked together and accordingly provided joint answers. The 27 users in fact produced 23 sets of answers to most questions.

However, some members of this set represent more than one application and in fact 44 individual applications were identified.

In conclusion then 44 sets of answers apply to questions 12 and 13 except part (c); 27 to the human questions 22, 23 and 25, and 23 to the remainder. In questions 15 and 18 the 23 sets of answers have been divided into those of 14 system programmers and 9 application programmers.

The 44 applications are distributed as follows:

Compilers	(3)
RT operating systems	(3)
Batch system	(1)
Systems programs - mainframe	(5)
Systems programs - mini	(8)
Numerical analysis	(2)
Applications - instrument control and analysis	(13)
Applications - on-line to production plant	(9)

Thus precisely half of the 44 applications are of a system program nature and half are user application programs.



13- For each application project state where possible:

- (a) Purpose of system (e.g. what does it control?)
- (b) Size of system (e.g. core store, peripherals)
- (c) Reasons for using the language
- (d) Proportion of total program in the system which was written in the language. State if this included code inserts and if so, how much?

CORAL 66

- (a) For our particular Modular 1: Four terminals, one operated remotely over a data-link, and a data-link to another computer.
- (b) For our particular Modular 1: 56k words of core, exchangeable disk-drive, line-printer, paper-tape peripherals.
- (c) For our particular Modular 1: Most convenient and efficient one available.
- (d) For our particular Modular 1: 100 per cent (less than 5 per cent in code inserts).

PAS 1 - PL/I

- (a) See 12
- (b) Core size from 16 to 32k words (16 bit)  
Peripherals: ASR, paper tape reader and punch, display, disk, type writer, line printer, process peripherals, (analog and digital I/O, counter, interrupt unit, clock).
- (c) Saving of time for learning the language, for programming, debugging and documentation.
- (d) Between 40 and 70%.

PROCOL

- (a) Not available.
- (b) 8k words to 24k words.
- (c) Best language available for this kind of applications efficiency of object code and specific language characteristics.
- (d) All the applications, including the tasking problems.

RTL/2

- (a) Details are confidential. See 12 for broad analysis.
- (b) 26 PDP/11 installations were analysed.

Core sizes were:

4k words	4 cases
8k words	8 cases
16k words	10 cases
24k words	2 cases
28k words	1 case
48k words	1 case

These 26 installations had the following peripherals in toto:

30 + teletypes

12	paper tape readers
9	paper tape punches
7	VDUs
3	RKO 5 disks
2	operator control panels
2	DEC tape units
1	64k disk
1	printer
1	MAG tape unit

plus assorted links to other machines.

The other 18 individual applications were system programs of a general nature running on IBM 360/370, ICL System 4 or general PDP/11s.

- (c) Reasons for using RTL/2 were given as follows:

Pleased with previous experience, accepted method, etc.	12
Best available	11
To interface with existing software written in RTL/2	7
Cheap, efficient, speed of writing, etc.	3

Specific language characteristics each mentioned once only were:

data definitions, re-entrancy, BYTES, structure, transportability.

RTL/2 (continued)

(d) This question is difficult to answer since most users do not write operating systems and do not know how much of the operating system part of their total system was written in code. Accordingly the question has been interpreted with reference to the software written by the individual users.

Four types of program are identified:

- i) The support system for running programs under the OS or J operating system on the IBM 360/370 or ICL System 4.
- ii) User programs (including compilers) on the IBM 360/370 or ICL System 4.
- iii) The MTS operating system for running programs on the PDP/11. Typical configurations.
- iv) User programs on the PDP/11.

	Total program size bytes	% program in modules written in RTL/2	% code inserts in the modules written in RTL/2
i	32k	83	0.8
ii	247k	99.8	0.12
iii	10k	75	4.0
iv	481k	100	0.54

In class (iv) most of the code was in one application - see answer to 20. If this application is omitted then a typical figure of 0.12% code is obtained.

Note: Where 13 is answered for several projects, answers to succeeding questions must state clearly whether they are for each project individually, for a typical project (and if so which one), or common to all the projects.

14- How did availability of other languages on the machine affect the use of the language under consideration?

CORAL 66

Not at all.

PAS 1 - PL/I

The only alternative was assembly language.

PROCOL

There is also LPG 15, FORTRAN, LP 15, BASIC available on MITRA 15; these did not affect the use of PROCOL language.

RTL/2

In the majority of cases the only alternative language was an assembly language. In only one case where an alternative high level language existed did any interaction occur. The other language was BASIC and interaction occurred through common files.

15- State significant features of the language found to be particularly useful.

CORAL 66

Most of the significant features of the language are useful.

PAS 1 - PL/I

Cannot be answered because analysis difficult.

PROCOL

Task handling, timers, interrupts and events handling, bit and fraction of words; I/O: standard and specialized.

RTL/2

The following features were cited. The number of times mentioned under the two major categories, application programmers (max 9) and system programmers (max 14) are given.

	Sys.	App.	Total
Records	11	8	19
BYTE mode	8	4	12
References - particularly with records	7	4	11



RTL/2 (continued)

PROC variables	7	2	9
Bit manipulation	4	4	8
LET - particularly naming constants	3	4	7
Explicit style - gives confidence	3	3	6
Initialization of data	3	1	4
Brick structure - particularly absence of lexical procedure nesting	2	2	4
BLOCK for local variables	2	2	4
CODE - access to RTL/2 variables therein	2	2	4
Arrays of initialized PROC variables	2	1	3
Simplicity and legibility of text	2	1	3
Security	1	2	3
String style	2	1	3
Switch statement	1	2	3

At this level further analysis becomes difficult since what is a significant feature for some is accepted as obvious by others. However to complete the analysis, the following were mentioned twice:

arrays of ref array bytes, LABEL variables for error exists, IF, re-entrancy, speed of writing, LENGTH operator, comments, generality of expressions, an ENT data brick containing ref array variables initialized to arrays in a local brick, and the following were mentioned once:

INT, REAL, the absence of FORMAT statements, double length products, SVC data, dereferencing, mixed mode arithmetic, ::= operator, key-words, AND/OR, the absence of input/output statements, ELSEIF...END construction, repetition factors in initialization, null arrays, overlapping EXT/ENT descriptions.

The only major difference between the two classes of users is that the PROC variable is much more significant to the system programmers.



16- In what way did the language assist the user in

- (a) Overall system design?
- (b) Program debugging?
- (c) Overall system commissioning?
- (d) Subsequent modification and maintenance?

CORAL 66

- (a) Block structure helped segmentation of design.
- (b) Most programming errors were trapped at compile-time.
- (c) Segmentation and self-documentation.
- (d) Segmentation and self-documentation.

PAS 1 - PL/I

All user-needed features are contained in the language.

PROCOL

- (a) The tasking structure and synchronization
  - The use of external subroutines
  - The use of timers and software events
  - The data description possibilities.
- (b) The debugging of programs made by other users was found easy.
- (c) No comment.
- (d) No comment.

RTL/2

- (a) The following features were mentioned, the number of times indicated. (Max 23).

The record structure	8
The data brick - especially its ability to control access to common data	8
Bricks generally	5
Modules	4
Procedure mechanism	4
Explicit EXT/ENT description	1
Re-entrancy	1

RTL/2 (continued)

One user stated that he was more ambitious than in his design since he was confident it would work.

One user specifically cited the use of initialized arrays of PROC variables for sequencing.

Two users stated that the language did not help in design and two didn't know.

- (b) This question was interpreted to mean syntactic debugging.

Users were satisfied, and on the whole found debugging RTL/2 programs easier, or much easier than debugging FORTRAN programs at this level. This was generally attributed to the lack of ad-hoc rules, the absence of default attributes and the inability to write ambiguous code. Of the 23 users only 3 had any adverse criticisms. These were minor ones concerning the compilers report of secondary errors and the position of errors.

- (c) This question was interpreted to mean routine debugging.

Of the 23 users, one made no comment and one had not tried to run his program yet. Of the remainder, 9 explicitly said commissioning was easy or very easy and only one said it was not easy.

The ease of commissioning was generally attributed to the predictability of the language and the use of records.

It was also stated that the records made it easy to write one's own debug package.

The run time debug packages on the mainframe machines were considered adequate.

Debugging on the PDP/11, where aids are provided, was satisfactory but the compiler produced assembly listing was considered vital. Although it was easy to find the few bugs and patch them, there was some danger with the cross compilation technique that the original source would not be updated but the patched program used permanently.

- (d) 5 of the 23 users felt unable to comment because their programs were under continuous development. Of the remainder the general feeling was that it was easy, although two users

remarked that relinking could be tedious. Specific language features mentioned which contributed to the ease were: (max 18)  
The general clear nature of the text with reserved key-words etc. 6  
The 'flat' procedure structure with procedures not nested made the general structure of the program much easier to follow than ALGOL programs 3  
Comments 2  
LET, modularity, REF arrays, absence of defaults, subroutines 1 each

- 17- State any language features which were found to be particular sources of error on the part of the user. State the attributed reason for these errors e.g. complexity of features, ambiguity, unfamiliarity.

CORAL 66

Anonymous references which allow reference to make to any location in store, because there is no compile-time check on the addresses referred to by them.

PAS 1 - PL/I

Cannot be answered.

PROCOL

None.

RTL/2

Errors can be classified into two sorts - those which are detected by the compiler and those which result in a runnable program which does not do what the programmer intended. The former can largely be neglected since their removal is a purely mechanical process. The latter are analysed below. The number of times mentioned is followed by the attributed cause.

RTL/2 (continued)

Omitting VAL when assigning indirectly via reference variables	6	ambiguity
Precedence of operators, particularly LAND/LOR/NEV	4	complexity
Precedence of AND/OR in conditions; unable to insert brackets to override/ensure precedence	3	ambiguity
Type conversion, particularly with fractions and reals	3	complexity
Double length truncation	2	complexity
Distinction between = and :=:	2	complexity
Confuse real and fraction constants	1	ambiguity

One user also mentioned that if new declarations were added to a data brick in different places in different modules then the linker may find no type clashes but nevertheless the program will be wrong.

It should be added that omitting VAL does not always lead to run time errors. In fact in most cases a compile time error will result. However, in view of the fact that dereferencing appears to be the greatest potential source of errors, it is clear that the coercion aspects of ALGOL 68 are rather bad aspects of the design of that language and should not be followed in future.

As far as compile time errors are concerned, no single feature was mentioned more than twice. Most were trivial such as omitting semi-colons. Only two users mentioned using reserved words as identifiers and both hastily added that they preferred reserved words and consequent legibility of the text.

- 18- State any language features which have not been used, or appear to be of little value to the applications described.

CORAL 66

Recursion is of use mainly to the compiler writer only.



PAS 1 - PL/I

Cannot be answered.

PROCOL

None.

RTL/2

The following features were cited. The number of times mentioned under the two major categories, application programmers (max 9) and system programmers (max 14) are given:

	Sys.	App.	Total
FRAC mode	10	8	18
Multidimensional arrays	11	4	15
REF LABEL/PROC/STACK mode	8	6	14
STACK	9	4	13
REAL mode	9	0	9
Arithmetic shifts	5	4	9
NEV operator	4	5	9
HEX constants	1	4	5
BIN constants	3	1	4
ABS operator	4	0	4
SHL operator	3	1	4
BLOCK structure	0	3	3
SHA operator	2	1	3
CODE statement	3	0	3
Record structures	2	1	3
NOT operator	2	1	3
Switch statement	3	0	3

At this level (as for question 4.4) further analysis becomes difficult since personal taste becomes very relevant. However, for the record the following were mentioned twice:

VAL, MOD operator, PROC mode, LENGTH operator, TITLE, WHILE statement, LABEL mode, division operator,

and the following were mentioned once:



RTL/2 (continued)

BY in for statement,  $::$  and  $:\neq$  operators, OPTION, LAND and LOR operators, logical shifts, double length, BYTE, multiplication operator.

The only significant difference between application and system programs is that the real mode is not useful in system programs.

19- State any "tricks" found to be necessary or convenient when using the language.

CORAL 66

It is necessary to know how the compiler stores strings to do anything but output them; code has to be used to obtain the starting addresses of procedures.

PAS 1 - PL/I

Cannot be answered.

PROCOL

None.

RTL/2

Of the 23 users, 16 used no tricks.

The trick of re-using stack space with BLOCK.....ENDBLOCK statements so that variables of different mode are equivalenced was mentioned 5 times. The following types of mode transfers using this trick were mentioned:

INT  $\longleftrightarrow$  FRAC

INT  $\longleftrightarrow$  REF INT

INT  $\longleftrightarrow$  REF plain

REAL  $\longleftrightarrow$  INT, INT

The dynamic creation of an array from a larger array using this trick was also cited once.

Two users had mapped different procedure descriptions onto the same procedure by tricking the linker.

State any situations which necessitate the use of code inserts.

CORAL 66

system calls in user programs; peripheral handling in the Operating System.

PAS 1 - PL/I

ne.

PROCOL

experience; especially with the way of handling I/O there is need to use code inserts.

RTL/2

the 23 users, 7 used no code.  
most uses were very small sequences - no more than 20 bytes each.  
these were:

access I/O devices directly, e.g. peripherals not	
handled by OS, specific plant inputs, handkeys	8
handle additional interrupts, startup, etc.	5
map variables of different mode (see also 19)	4

the transfers specifically mentioned were:

REAL — ARRAY (8) BYTE  
REF INT — INT  
ARRAY () BYTE — ARRAY () INT

call existing loaders in a host environment	2
copy whole records	1
ensure adequate stack space for slave subroutines	
that system could not have an unrecoverable error in a	
critical situation. In this case the code statement did not	
generate instructions but acted as a communication channel	
the compiler	1
overcome a compiler imposed implementation restriction	
array lengths	1
create global arrays out of a core area dynamically	1

RTL/2 (continued)

To create a local array (for re-entrant use)	1
To construct hidden parameters of procedures (like POP-2 partial application)	1

Four uses of code were of a more extensive nature. These were:

In one application on a small machine (8k) core was particularly short and about 1k words of code was written in DDC routines. Savings of space were particularly attributed to obtaining both quotient and remainder on division in one instruction and double length overflow detection.

In other application one interrupt routine was entirely hand coded to avoid the timing overhead of a procedure call in a loop. This was 150 bytes of code.

A square root routine was written in code to obtain a large increase in speed. This was 50 bytes of code. Note that all other standard function routines are written in RTL/2.

Finally some modules of an operating system which were written in RTL/2 were also hand coded - partly to save space on small machines but partly because spare effort was available at the time. This was about 3k bytes of code. There is some doubt as to whether this was worthwhile.

21- Have programs written in the language been successfully transported from one machine type to another? If so, what effort was involved?

CORAL 66

Yes. Extremely variable, from 1 man-day upwards, depending on the care taken when originally writing the program.

PAS 1 - PL/I

The compilers for PAS 1 and PL/I-subset have been transported from IBM/360 to Honeywell H316/DDP516.

PROCOL

Not yet achieved.

RTL/2

Yes, 7 users had done this, mostly from IBM 370 or ICL System 4 to PDP/11. In all cases no effort other than the purely mechanical effort of calling a different compiler was involved. The programs were completely unchanged.

22- What sort of formal training is necessary for the language user? Answers should cover:

- (a) Conversion of the experienced high-level language programmer.
- (b) Conversion of the experienced assembly language programmer.
- (c) Training of those people who had no previous programming experience.
- (d) Conversion of those experienced in both high-level and assembly languages.

CORAL 66

- (a) Very roughly, one weeks training, preferably hands-on.
- (b) Very roughly, two weeks.
- (c) Four weeks given a reasonable aptitude for programming.
- (d) (\*).

PAS 1 - PL/I

Between 2 and 5 days.

PROCOL

Real time features and I/O handling are of some importance in the training.

- (a) 4 days.
- (b) 6 days.
- (c) 7-8 days.
- (d) 3 days.



RTL/2

4 users refused to answer this question on the grounds that they had 'grown up' with the language. Of the remainder, most only answered the category appropriate to themselves. Average values in terms of length of formal course were:

- (a) 4 1/2 days.
- (b) 4 days.
- (c) 5 days.
- (d) 3 1/2 days.

23- For each of 22 (a), (b), (c) and (d) state the time required after formal training for the programmer to become proficient in use of the language.

CORAL 66

Depends so much on the individual; possible estimate 6-12 months.

PAS 1 - PL/I

Cannot be answered.

PROCOL

Impossible to answer.

RTL/2

In general, users felt unable to answer this question. Where figures were given they ranged from 'days' to '3 months'. An interesting answer was '3k words of generated code'.

24- State any language features which were noticeably absent and why they were needed.

CORAL 66

(\*)



PAS 1 - PL/I

Extensions of possibilities for tasking and scheduling, use of "two" languages namely PAS 1 and PL/I-subset was felt to be uncomfortable.

PROCOL

Pointer variables.

RTL/2

5 users said there were no features noticeably absent. A few features were mentioned more than once but the majority of features mentioned were only mentioned once and those mostly by only a few (3 or 4) users.

Some sort of ability to manipulate general references, permit mode transfers, map structures, etc. One user wished for mode tags so that modes could be interrogated - the other 11 wished for purely anonymous references	12
Ability to trim arrays for more flexible storage control	3
Logical operations on fractions - particularly to mask out junk bits on input	3
Mandatory overflow detection	3
Static length local arrays of plain mode for greater re-entrancy	3
Unsigned arithmetic	2
Whole array and record copying for greater efficiency	2

The following were mentioned once only:

NOT on bytes, case expression, cyclic shift, REF REF mode, conditional operators ( in writing and interpreter), user defined operators, intermediate double length storage, local label variables for efficient private control (value local to current procedure only), ability to mix statements/declarations before first label, etc., records within records, conditional destination

RTL/2 (continued)

in assignment statements, obtain quotient and remainder in one operation, exit from loop without use of a label, local records, compile time arithmetic on constant expressions (especially in LET definitions), ability to describe stack detail to compiler, named fields, cram setting of constants, input-output, double length addition.

One user mentioned particular problems in the use of fractions. Main need was to do 'limiting arithmetic' that is to do an operation and replace by maximum value if overflow occurs. This is a particular problem with multiplication since the double length product cannot be decomposed into its two parts without recomputation.

25- What is felt to be:

- (a) The best feature of the language?
- (b) The worst feature of the language?
- (c) The most obvious omission from the language?

CORAL 66

- (a) The efficiency of many of its compilers and the support provided by RRE and NCC.
- (b) Probably the ability to manipulate machine addresses via anonymous references and the LOCATION operator.
- (c) Explicit tasking and I/O features and the limited ALGOL 60 type data structures.

PAS 1 - PL/I

- (a) Ability of writing the whole program in high level language.
- (b) Use of two languages (see 24).
- (c) Some tasking and scheduling possibilities.

PROCOL

- (a) Real time features, I/O for specialized peripheral, Fortran-likeness.
- (b) None.
- (c) Pointers variables.

RTL/2

The answers to this question are not easy to analyse, mainly because similar concepts are expressed in different ways. Also many answers did not refer to the language itself but to aspects of associated operating systems, etc.

- (a) Overall clarity of programs      5  
Records                                      4

Other features mentioned once or twice were:

machine independence, re-entrancy, flexibility, user oriented security, modularity, ease of writing, instiles confidence, compactness of definition, cohesion, freedom of expression, references.

- (b) Most complaints were about some aspect of system generation, compilation, etc. No single language feature was mentioned more than once.

Those mentioned just once were:

precedence rules, fixed point arithmetic, typing ("#NL#. power obscures other programs, big at first sight, refs hard to explain, options, no security on overlays, too secure, dereferencing, new.

- (c) Some form of general reference

Other omissions mentioned only once were:

trimmable arrays, double length, fraction overflow detection, input-output.

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, VA 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2 copies
Office of Naval Research Code 102IP Arlington, VA 22217	6 copies
Office of Naval Research Code 200 Arlington, VA 22217	1 copy
Office of Naval Research Code 455 Arlington, VA 22217	1 copy
Office of Naval Research Code 458 Arlington, VA 22217	1 copy
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1 copy
New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D. C. 20375	6 copies



OFFICIAL DISTRIBUTION LIST (Cont.)

Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D. C. 20380	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, CA 92152	1 copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, MD 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D. C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D. C. 20350	1 copy